Efficient Implementation of Elliptic Curve Cryptography in Software: Protocols and Improved Scalar Multiplication Methods on Koblitz Curves

Christian Wagner

Institute for Applied Information
Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a
8010 Graz, Austria

Master's Thesis

Supervisors:
Dipl.-Ing. Christian Hanser
Dipl.-Ing. Dr.techn. Michael Hutter

Assessor:
Univ.-Prof. Dipl.-Ing. Dr.techn. Roderick Bloem

May 2014

Statutory declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ____________________ ______________________________

(date) (signature)

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ________________ ______________________________

(Datum) (Unterschrift)

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Acknowledgements

First, I would like to thank Roderick Bloem for serving as the assessor of this thesis. Second, I would like to offer my special thanks to Christian Hanser and Michael Hutter, my supervisors, who gave me an understanding of elliptic curve theory and the principles of academic work. Moreover, I would like to thank them for their support and patience during the development of this thesis and the efforts they spent on corrections and remarks.

I would also like to sincerely thank my parents who have unconditionally supported me in every possible way. Furthermore, I am also very thankful to my love Stephanie for always being there for me and encouraging me whenever I needed it. Finally, I would like to thank my friend Gerwin for the companionship throughout all these years of studying.

Abstract

These days, elliptic curves are widely used for security-related applications that provide cryptographic services such as key agreement and digital signature generation and verification. All of these applications need scalar multiplication, which is the most expensive arithmetical operation on elliptic curves.

In this master's thesis, we discuss security properties of elliptic curves and examine some attacks against the elliptic curve discrete logarithm problem. We present several protocols which are based on elliptic curves, such as the Elliptic Curve Digital Signature Algorithm (ECDSA) and the Elliptic Curve Integrated Encryption Scheme (ECIES). In these protocols, scalar multiplication is an essential element, hence we also detail various methods for scalar multiplication on elliptic curves. Within this context, we introduce a modification to the fixed-base comb multiplication method. When it comes to speed regarding scalar multiplication, the fixed-base comb multiplication method, which is due to Lim and Lee, is one of the preferable methods. The presented modification exploits the possibility of exchanging doubling operations with much cheaper applications of the Frobenius endomorphism on binary Koblitz curves. We have implemented this modification in software and compare its performance with the performance of the windowed τ-adic non-adjacent form (WTNAF) multiplication method implementation and the performance of the conventional fixed-base comb method. The comparison of a single scalar multiplication shows a performance improvement of up to 25% over the WTNAF method and an improvement of up to 42% compared to the conventional comb method. We show that not much effort is required for the implementation of the τ-adic comb method and that it is a good alternative to other fixed-base multiplication methods.

Keywords: elliptic curves, ECC, Pollard's rho, SSSA, MOV, ECDSA, ECIES, ECDH, ECMQV, scalar multiplication, Lim-Lee method, fixed-base comb method, Koblitz curves, Frobenius endomorphism, WTNAF, τ-adic representation

Kurzfassung

Heutzutage sind elliptische Kurven weitverbreitet für sicherheitsrelevante Anwendungen, die kryptographische Services, wie zum Beispiel Schlüsselaustausch und Generierung und Verifikation digitaler Signaturen, bereitstellen. All diese Anwendungen erfordern die Skalarmultiplikation, welche die aufwendigste arithmetische Operation auf elliptischen Kurven ist.

In dieser Masterarbeit betrachten wir die Sicherheitseigenschaften von elliptischen Kurven und untersuchen einige Angriffe auf das Diskrete-Logarithmus-Problem in elliptischen Kurven. Wir stellen auch mehrere Protokolle vor, die auf elliptischen Kurven basieren, wie zum Beispiel den Elliptic Curve Digital Signature Algorithm (ECDSA) und das Elliptic Curve Integrated Encryption Scheme (ECIES). In diesen Protokollen ist die Skalarmultiplikation ein grundlegendes Element, daher zeigen wir auch verschiedene Methoden für Scalarmultiplikation auf elliptischen Kurven. In diesem Zusammenhang präsentieren wir eine Modifikation der Fixed-base Comb Multiplikationsmethode. Wenn es um Geschwindigkeit bei der Skalarmultiplikation geht, ist die Fixed-base Comb Multiplikationsmethode von Lim und Lee eine der bevorzugten Methoden. Die vorgestellte Modifikation nutzt die Möglichkeit, auf binären Koblitz-Kurven Doubling-Operationen gegen viel weniger aufwendige Anwendungen des Frobenius-Endomorphismus auszutauschen. Wir haben diese Modifikation in Software implementiert und vergleichen ihre Leistung mit der Leistung der Implementierung der Windowed τ-adic non-adjacent form (WTNAF) Multiplikationsmethode und der Leistung der herkömmlichen Fixed-base Comb Methode. Der Vergleich einer einzelnen Skalarmultiplikation zeigt eine Leistungssteigerung von bis zu 25% gegenüber der WTNAF-Methode und von bis zu 42% gegenüber der herkömmlichen Comb Methode. Wir zeigen, dass die Implementierung der τ-adischen Comb Methode nur wenig Aufwand erfordert und dass sie eine gute Alternative zu anderen Fixed-base Multiplikationsmethoden ist.

Stichwörter: elliptische Kurven, ECC, Pollard's rho, SSSA, MOV, ECDSA, ECIES, ECDH, ECMQV, Skalarmultiplikation, Lim-Lee Methode, Fixed-base Comb Methode, Koblitz-Kurven, Frobenius-Endomorphismus, WTNAF, τ-adische Darstellung

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

In 1985, Neal Koblitz [Kob87] and Victor S. Miller [Mil85] independently discovered a way of using elliptic curves over finite fields for public key cryptography. Both proposed to use the group of points on an elliptic curve defined over a finite field in a discrete logarithm cryptosystem. Nowadays, elliptic curve cryptography (ECC) systems are widely used for security related applications like key agreement and digital signature generation and verification. The mathematical basis of the security of ECC is the intractability of the elliptic curve discrete logarithm problem (ECDLP). In contrast to RSA, there is no algorithm known that can solve the ECDLP in subexponential time on today's computers, provided that the curve and the underlying finite field are chosen properly. As a consequence, it is assumed that the ECDLP is significantly harder than the integer factorization problem (IFP), which is the basis of other cryptosystems like RSA. Therefore, ECC systems have a greater strength-per-keybit compared to RSA and, thus, require smaller parameters to offer approximately the same level of security. For example, an encryption with a 224 bit elliptic curve key equals an encryption with a 2048 bit RSA key in terms of security. A comprehensive list of comparable security strengths is available in [BBB$^+$12]. The use of smaller parameters results in faster computations and smaller keys, signatures and certificates. This is quite important in environments with limited processing power, low memory resources or constraint power consumption and bandwidth, e.g. smart cards, cell phones or other embedded systems.

For a long time there were no important ECC-based protocols that made use of the special structures of elliptic curves. The protocols used were simply straightforward adaptions of the ones developed earlier for finite fields. This changed around the year 2000, when pairing-based cryptosystems emerged ([Jou00], [BF01], [BLS01], [Sma01]). The idea of bilinear pairings of algebraic curves was not really new. The Weil pairing and the Tate pairing had already been used for attacks against the ECDLP. However, the whole potential of pairings had not been recognized until then. Pairings offer a way to implement schemes where no other efficient implementation is known, such as identity-based encryption or attribute-based encryption. In the last few years there has been a lot of research regarding efficient implementation of bilinear pairings and a large number of cryptographic primitives based on bilinear mappings have been introduced. Based on these primitives, numerous ECC-based schemes have been developed providing identity-based encryption as well as key agreement, pairing-based signatures and signcryption.

Over the years the trust in ECC was shaken to the very foundations several times. The first time this happened was in 1991, when the MOV-attack was published. It provided an efficient computation of the discrete logarithm on the so-called supersingular elliptic curves. This type of curves was very popular then, because it provided fast computations. In 1994, Adleman et al. presented a subexponential-time algorithm for the discrete logarithm problem on hyperelliptic curves of a genus greater than 4 [ADH94]. Before, hyperelliptic curve cryptography, proposed by Neil Koblitz in [Kob89], had been thought to be at least as secure as elliptic curves. Although the proposed algorithm did not affect elliptic curves themselves, which are in fact hyperelliptic curves of genus 1, it gave reason to question the security of ECC in general. ECC suffered from another setback in 1997, when the SSSA attack was published, which gave a polynomial-time algorithm to solve the ECDLP on prime-field anomalous curves.

Recently, in 2013, Antoine Joux claimed that he had set a new speed record for the computation of discrete logarithms in finite fields [Jou13b]. He was able to compute the discrete logarithms on supersingular curves of characteristic 2 in reasonable time using the index calculus algorithm he proposed in [Jou13a]. Thereby he showed that these curves are not suitable for secure pairing-based cryptography. Based on Joux' algorithmic ideas, several computation records for the discrete logarithm on supersingular curves of characteristic 2 and 3 have been set [Gal14]. As a consequence, symmetric pairings, often called Type-1 pairings [GPS08], can be considered unusable in terms of security. However, a lot of ECC-based protocols have been designed for Type-1 pairings only.

In 2006, the trust in the standardization procedure and the recommended curve parameters of ECC of the National Institute of Standards and Technology (NIST) was shaken when it was discovered that the ECC-based pseudorandom number generator Dual Elliptic Curve Deterministic Random Bit Generator (`Dual_EC_DRBG`) [BK12], which had been claimed to be cryptographically secure, had a serious weakness. The author of [Bro06] as well as the authors of [SS06] showed that the output of the random number generator can be empirically distinguished from random bits. This flaw was interpreted as a sign of a possible backdoor in the algorithm. There is evidence that despite this flaw, or perhaps because of it, the `Dual_EC_DRBG` had been included in the NIST standard on behalf of the NSA. In connection with this exposure, security experts currently question the choice of parameters of the elliptic curves suggested by the NIST standard, although there has been no evidence for weaknesses so far. Some experts, however, suggest using Brainpool curves [LM10] or the `Curve25519` elliptic curve proposed by Bernstein in [Ber06], which is designed as a very fast, high-security elliptic curve Diffie-Hellman function, instead of the NIST-recommended parameters.

Irrespective of the employed curves, scalar multiplication is the most expensive operation in ECC when it comes to computational effort. Consequently, improving multiplication methods is essential for the performance of cryptosystems on elliptic curves. One way to increase the performance of scalar multiplications is to exploit the curve's structure, such as the structure of Koblitz curves. These curves are equipped with the so-called Frobenius endomorphism which can be used to achieve tremendous speedups for scalar multiplications. In the context of Koblitz curves, many multiplication methods relying on a non-adjacent form (NAF) have been adapted so that they exploit the available endomorphism. However, until now there has been no such modification to the fixed-base comb multiplication method, which is one of the fastest scalar multiplication methods available.

1.1 Contribution

In this master's thesis, we are going to present an improvement of the fixed-base comb methods on Koblitz curves that we proposed in [HW13]. We show a scalar recoding to obtain an unsigned τ-adic representation of scalars, which provides the possibility to exchange the point doublings within the comb methods for applications of the Frobenius endomorphism. We detail the modifications to the fixed-base comb methods that are required in order to gain a performance benefit from the scalar recoding. Furthermore, we have implemented the modified comb method as well as the conventional fixed-base comb method and the WTNAF method in software. We give a detailed performance comparison and illustrate the performance gains we have achieved with respect to the WTNAF and the conventional fixed-base comb methods. Especially the comparison with the WTNAF multiplication method is of great interest, as it exploits the Frobenius endomorphism on Koblitz curves as well. We achieve performance improvements of up to 25% compared to the WTNAF method and performance improvements of up to 42% compared to the conventional fixed-base comb method.

1.2 Outline

At the beginning of this thesis, Chapter 2 gives an overview of the mathematical background and fundamental concepts of elliptic curves. Based on these preliminaries, Chapter 3 introduces the reader to the security properties of elliptic curves and details several state-of-the-art attacks against the ECDLP. Chapter 4 deals with protocols and cryptographic schemes on elliptic curves. It covers digital signatures and key-agreement schemes, as well as a hybrid encryption scheme. Chapter 5 is devoted to scalar multiplication methods on elliptic curves. It discusses three different types: basic algorithms, fixed-point algorithms, and multiple point multiplication algorithms. In Chapter 6, a comprehensive overview of Koblitz curves is given. Chapter 7 introduces a recoding of scalars, the so-called unsigned τ-adic representation. We detail how it can be efficiently obtained from an integer and we present a modification to the fixed-base comb method, discussed in general in Chapter 5, which offers a performance benefit on Koblitz curves from using this representation. In this context, we have implemented this modified method in software and compare its performance to the performance of the implementations of conventional algorithms. Finally, this thesis is concluded in Chapter 8.

Chapter 2

Preliminaries

This chapter is intended as an overview of the fundamental mathematical principles that are essential for the subsequent parts of this thesis. It treats the algebraic concepts of groups, rings and fields as well as the fundamentals of elliptic curves. Readers who are already familiar with these topics can safely skip this chapter.

The chapter is composed of four sections. Section 2.1 is devoted to the algebraic background of elliptic curve cryptography. It covers introductions to rings, groups, and fields, as well as an introduction to the discrete logarithm problem, the Freshman's dream, and Lucas sequences. Section 2.2 briefly discusses binary fields and the corresponding arithmetic. The third section, Section 2.3, provides an introduction to elliptic curves in general as well as an insight into point representation in form of projective coordinate types. At the end, the chapter is briefly summarized in Section 2.4.

This chapter is primarily based on the books [HMV04], [Sil09], [Her96], [DF04], [Wal98], and [Sma12]. Additional information is taken from [Rib91], [Han10], [CLO07], [Rib00] and [JMV01], as well as from the lecture notes [Die07] and [Die09].

2.1 Algebraic Background

This section gives a short introduction to the essential algebraic structures as well as some corresponding concepts. These topics form the basis of public key cryptography in general.

2.1.1 Groups

A group $(\mathcal{G}, \cdot)$ is a non-empty set $\mathcal{G}$ with an arithmetic operation $\cdot$ that satisfies the following properties:

- $\mathcal{G}$ is closed under the operation $\cdot$, i.e., if $a, b \in \mathcal{G}$ then also $a \cdot b \in \mathcal{G}$.
- The associative law holds in $\mathcal{G}$. For all elements $a, b, c \in \mathcal{G}$ it applies that $(a \cdot b) \cdot c = a \cdot (c \cdot b)$.
- A special element $e \in \mathcal{G}$ exists such that $a \cdot e = e \cdot a = a, \forall a \in \mathcal{G}$. This element e is called the *identity element* of $\mathcal{G}$.
- For every $a \in \mathcal{G}$ there exists an inverse element, denoted by a^{-1}, such that $a^{-1} \cdot a = a \cdot a^{-1} = e$.

A group $(\mathcal{G}, \cdot)$ is called *Abelian* if it is commutative, i.e. $a \cdot b = b \cdot a$ for all $a, b \in \mathcal{G}$.

Note that groups can also be written in an additive way with $+$ as group operation. If a group $\mathcal{G}$ contains a finite number of elements then $\mathcal{G}$ is called *finite* and *infinite* otherwise. The number of elements (*cardinality*) of a finite group $\mathcal{G}$ is called the *order* of $\mathcal{G}$, denoted by $|\mathcal{G}|$ or $ord(\mathcal{G})$.

A group $\mathcal{G}$ that contains an element g from which every other element $y \in \mathcal{G}$ can be obtained by repeatedly applying the group operation, is called *cyclic*. In the case of a cyclic multiplicative group $(\mathcal{G}, \cdot)$, every element $y \in \mathcal{G}$ can be expressed as

$$y = \underbrace{g \cdot g \cdot \ldots \cdot g}_{x \text{ times}} = g^x \text{ with } x \in \mathbb{Z}.$$

This element g is called a *generator* of the group $\mathcal{G}$. A group generated by some generator g is often denoted by $\langle g \rangle$. A group can have multiple generators.

Example 2.1.1 (Generators)**.** Let $(\mathbb{Z}_5^* = \{1, 2, 3, 4\}, \cdot)$ denote a multiplicative, Abelian group modulo 5 with neutral element 1. The generators of $(\mathbb{Z}_5^*, \cdot)$ are the elements 2 and 3:

$$\begin{aligned} 2^1 &= 2 & 3^1 &= 3 \\ 2^2 &= 4, & 3^2 &= 9 \equiv 4 \bmod 5 \\ 2^3 &= 8 \equiv 3 \bmod 5 & 3^3 &= 27 \equiv 2 \bmod 5 \\ 2^4 &= 16 \equiv 1 \bmod 5 & 3^4 &= 81 \equiv 1 \bmod 5 \end{aligned}$$

The order $ord_{\mathcal{G}}(x)$ of an element $y \in \mathcal{G}$ is the smallest positive integer x, such that $y^x = e$ Thus, the order of the generators in a group is equal to the order of the group. If no such x exists, the element y has infinite order.

Discrete Logarithm Problem

In a cyclic group $\mathcal{G}$ of order n, the smallest positive integer x for which it applies that $y = g^x$, where $y \in G$ and g is a generator of $\mathcal{G}$, is called the *discrete logarithm* (DL) of y to the base g. The DL is the analogue to the ordinary logarithm with regards to multiplicative cyclic groups. Thus, important features of the ordinary logarithm remain valid for the DL:

- $\log_g(y \cdot z) = \log_g(y) + \log_g(z) \mod n$,
- $\log_g(y^a) = a \log_g(y) \mod n$, and
- $\log_h(y) = \log_g(y) \cdot \log_g(h) \mod n$,

where g and h are different generators of $\mathcal{G}$, $y, z \in \mathcal{G}$ and a is an integer.

The security assumptions of many cryptographic schemes rely on the intractability of the *discrete logarithm problem* (DLP).

Definition 2.1 (DLP)**.** Given a cyclic group $(\mathcal{G}, \cdot)$ of order n with a generator g, and an element $y \in \mathcal{G}$, then the DLP is the difficulty to find the DL to the base g of y, i.e., finding an integer x such that:

$$y = g^x.$$

The difficulty of solving the DLP depends on the group structure. For example:

- In an additive cyclic group $(\mathbb{Z}_p^*, +)$, the DLP is easy to solve, since, if $g \in \mathbb{Z}_p^*$ is the generator, the problem to solve is $y \equiv xg \mod p$.
- In a group of the shape $(\mathbb{Z}_p^*, \cdot)$, where p is prime, the DLP is hard if $ord(\mathbb{Z}_p^*)$ comprises a large prime factor q, i.e., $q \mid p-1$. In this context, such prime p is often called *safe prime*. However, even if p is a safe prime, there exists an algorithm that can find the DL in subexponential time (see "Index Calculus" in Section 3.4.2). Therefore, in this case p is additionally required to be very large.

Diffie-Hellman Problem

The Diffie-Hellman problem (DHP) is closely related to the DLP. Its intractability is the basis for various public-key cryptosystems, e.g., the Diffie-Hellman key agreement and the ElGamal public-key encryption scheme.

Definition 2.2 (DHP)**.** Given a generator g of $\mathcal{G}$, and two elements g^a and g^b, then the DHP is the problem of finding an element y such that:

$$y = g^{ab}.$$

Note that the DHP is not harder than the DLP since, if the DLP can be efficiently solved for $y = g^x$, then so can the DHP.

Homomorphisms

A homomorphism in abstract algebra is a structure-preserving map between two algebraic structures. More precisely, a homomorphism between two groups $(\mathcal{G}, \cdot)$ and $(\mathcal{G}', *)$ is a map

$$\varphi : \mathcal{G} \to \mathcal{G}'$$

that ensures that

$$\varphi(ab) = \varphi(a)\varphi(b) \tag{2.1}$$

holds for all $a, b \in \mathcal{G}$. Note that in Equation (2.1), the product on the left side is computed in $\mathcal{G}$ and the product $\varphi(a)\varphi(b)$ on the right side is computed in $\mathcal{G}'$. The homomorphism preserves the group operation as well as the identity element (i.e. $\varphi(e_{\mathcal{G}}) = e'_{\mathcal{G}}$) and the inverse (i.e. $\varphi(a)^{-1} = \varphi(a^{-1})$). A homomorphism $\varphi : \mathcal{G} \to \mathcal{G}'$ is called

- *monomorphism* if φ is one-to-one (injective),
- *epimorphism* if φ is onto (surjective),
- *isomorphism* if φ is a monomorphism that is onto (bijective),
- *endomorphism* if $\mathcal{G}$ and $\mathcal{G}'$ are the same, and
- *automorphism* if φ is an isomorphism and also an endomorphism.

Two groups $\mathcal{G}$ and $\mathcal{G}'$ are said to be *isomorphic* if there is an isomorphism of $\mathcal{G}$ onto $\mathcal{G}'$ (denoted as $\mathcal{G} \simeq \mathcal{G}'$).

2.1.2 Rings

Definition 2.3. A *ring* is a non-empty set R with two binary operations $+$ and $\cdot$. For a ring $(R, +, \cdot)$ the following properties apply:

(i) $(R, +)$ is an Abelian group with identity element 0_R.

(ii) Multiplication is associative: $\forall a, b, c \in R : (a \cdot b) \cdot c = a \cdot (b \cdot c)$.

(iii) Multiplicative identity: $\exists 1_R \in R : \forall a \in R : 1_R \cdot a = a \cdot 1_R = a$.

(iv) The two distributive laws

- $a \cdot (b + c) = a \cdot b + a \cdot c$,
- $(a + b) \cdot c = a \cdot c + b \cdot c$

hold for all $a, b, c \in R$.

2.1.3 Fields

A field $(K, +, \cdot)$ is a commutative ring such that additionally:

(i) $(K, +)$ is an (additive) Abelian group with identity element 0_K.

(ii) $(K^*, \cdot)$ is a (multiplicative) Abelian group with identity element 1_k, where $K^* = K \setminus 0_K$.

(iii) The distributive laws hold for both groups.

In a field every nonzero element is invertible. A field is said to be *finite* if it has only a finite number of elements.

The *characteristic* of a field K, often *char(K)*, is the smallest positive integer p so that $p \cdot 1_K = 0$, where 1_K is the identity of the field K. If no such p exists, the characteristic is 0.

Field Extensions

Consider K to be a subset of field L. If K itself is a field under the operations of L, then K is a *subfield* of L. The field L is said to be an *extension field* of K, denoted L/K. If $K \neq L$, then K is called a *proper subfield* of L. For every prime power p^m there is up to isomorphisms exactly one finite field. This field has characteristic p and contains all subfields $\mathbb{F}_{p^l}$ for $l|m$.

2.1.4 Freshman's Dream

The Freshman's dream is the name of the common mistake $(x + y)^n = x^n + y^n$ that is made when computing the power of a sum of real numbers in a field of characteristic 0. It is obvious that in general this is incorrect. However, in fields with prime characteristic p, $(x + y)^n$ does evaluate to $x^n + y^n$.

Lemma 2.1 (Freshman's dream)**.** If x and y are elements of a field K, which has prime characteristic p, then

$$(x + y)^p = x^p + y^p.$$

Proof. $(x+y)^p$ written in binomial formula, we get:

$$(x+y)^p = \sum_{k=0}^{p} \binom{p}{k} x^{p-k} y^k = \sum_{k=0}^{p} \cdot \frac{p!}{k!(p-k)!} x^{p-k} y^k \tag{2.2}$$

Looking at the prime factors of the binomial coefficients $\frac{p!}{k!(p-k)!}$ in Equation (2.2), we can see that for all $0 < k < p$, p divides the term $\frac{p!}{k!(p-k)!}$ and due to the commutative ring, this term evaluates to 0. For the other two cases, the result of the fraction is 1. Thus, from Equation (2.2) we have two remaining terms that are nonzero:

$$(x^p y^0) + (x^0 y^p) = x^p + y^p.$$

□

2.1.5 Greatest Common Divisor

The *greatest common divisor* (GCD) is the greatest positive integer that divides two (or more) integers, which are not all 0, without remainder.

Definition 2.4 (GCD)**.** The greatest common divisor of $a, b \in \mathbb{Z}$ with $a \neq 0 \vee b \neq 0$ is the greatest positive $c \in N$ such that:

$$\gcd(a,b) = \max\{c \in \mathbb{N} : c \mid a \wedge c \mid b\}.$$

2.1.6 Lucas Sequences

The *Lucas sequences* $U_n(p,q)$ and $V_n(p,q)$ are sequences of integers characterized by two fixed integers p and q. The sequences satisfy the recurrent relation $x_n = qx_{n1} - qx_{n2}$. Every sequence that satisfies this relation can be expressed as a linear combination of $U_n(p,q)$ and $V_n(p,q)$.

Before we detail the Lucas sequences, we are going to briefly explain the *roots* and the *discriminant* of polynomials. The *roots*, also called *zeros*, of a polynomial $f(x)$ are all values for x such that $f(x) = 0$. The *discriminant* of a polynomial gives information about the nature of the polynomial's roots. For a quadratic polynomial $f(x) = ax^2 + bx + c$, the discriminant is $D = b^2 - 4ac$.

Now, consider the polynomial $x^2 - px + q$ where p and q are two nonzero integers. The nonzero discriminant D of the polynomial is $D = p^2 - 4q$ and its roots α and β are

$$\alpha = \frac{p + \sqrt{D}}{2} \quad \text{and} \quad \beta = \frac{p - \sqrt{D}}{2}.$$

For these roots it applies that $\alpha \neq \beta$, $\alpha + \beta = p$, $\alpha\beta = q$, and $(\alpha - \beta)^2 = D$.
Now, we define $U_n = U_n(p,q)$ and $V_n = V_n(p,q)$ for every $n \geq 0$:

$$U_0 = 0, \quad U_1 = 1, \quad U_n = pU_{n-1} - qU_{n-2},$$

$$V_0 = 2, \quad V_1 = p, \quad V_n = pV_{n-1} - qV_{n-2}.$$

The sequences U_n and V_n are called the first and the second *Lucas sequences with parameters* (p,q). V_n is also called the *companion* Lucas sequence with parameters (p,q). Special Lucas sequences (with $p = 1$, $q = -1$ and the corresponding $d = 5$) are

- Fibonacci numbers: $U_n = U_n(1, -1)$: 0, 1, 2, 3, 5, 8, 13, 21, ...
- Lucas numbers: $V_n = V_n(1, -1)$: 2, 1, 3, 4, 7, 11, 18, 29, ...

2.2 Binary Fields

A *binary field* is a finite field of order 2^m. These types of fields are often also called *characteristic-two finite fields.* For binary fields there are two common ways to represent field elements: the *polynomial basis* an the *normal basis.* In the following we focus solely on the polynomial basis.

A binary polynomial $f(x)$ is a polynomial whose coefficients are in $\mathbb{F}_2$. The set of all these polynomials is denoted as $\mathbb{F}_2[x]$.

The *degree* of a polynomial $f(x)$, denoted $deg(f)$, is given by the greatest index i where the coefficient a_i is nonzero. If there is no such index i, then $deg(f) = -\infty$. This polynomial is called *zero polynomial.*

Definition 2.5 (Irreducible Polynomial)**.** A polynomial $f(x) \in \mathbb{F}_2[x]$ is called *irreducible* if $deg(f) > 0$ and its only divisors are nonzero constant polynomials, i.e., $g(x) \mid f(x)$ where $g(x) \in \mathbb{F}_2$ or $g(x) = f(x)$.

Definition 2.6 (Monic)**.** A polynomial $f(x)$ in an arbitrary field is called *monic*, if its leading coefficient $a_n = 1$, i.e., a monic polynomial is of the form:

$$f(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_1x + a_0.$$

Let $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ be a monic, irreducible polynomial over $\mathbb{F}_2$ of degree m and let $\alpha \in \mathbb{F}_{2^m}$ be its root. Then, the polynomial basis of the field $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$. The polynomial $f(x)$ is referred to as *reduction polynomial.*

In polynomial basis representation, the elements of $\mathbb{F}_{2^m}$ are all the binary polynomials with a maximum degree of $m - 1$:

$$\mathbb{F}_{2^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_2x^2 + a_1x + a_0 : a_i \in 0, 1\}.$$

In polynomial basis representation, a field element $a_{m-1}x^{m-1} + \ldots + a_1x + a_0$ of $\mathbb{F}_{2^m}$ is represented by the bitstring $(a_{m-1}, \ldots, a_1, a_0)$.

2.2.1 Arithmetics

In order to simplify the notation, we are going to denote a binary polynomial $g(x)$ by g for the rest of this chapter. The *addition* and *subtraction* of field elements using a polynomial basis equals the common polynomial addition, with coefficient arithmetic modulo 2. Since $-1 = 1$ in characteristic 2, addition and subtraction are the same operation on binary fields.

The *multiplication* of field elements can be performed as common multiplication, with an additional *reduction modulo f.* This reduction is essential since the result can have a degree up to $2m - 2$. The reduction by the *reduction polynomial* f results in a unique remainder polynomial r of degree less than m. Various methods for such a reduction can be found in [HMV04, Section 2.3.5]. There are numerous techniques for speeding up the multiplication, e.g., the right-to-left comb method [HMV04, Algorithm 2.34] or the Karatsuba-Ofman algorithm [KO62], which can be directly adapted for polynomial multiplication.

Squaring of a field element can be carried out very efficiently since for binary polynomials, squaring is a linear operation. Let the binary representation of element a be $(a_{m-1}, a_{m-2}, \ldots, a_1, a_0)$ where $a_i \in \mathbb{F}_2$. In order to obtain a^2 there is a 0 inserted between each two consecutive bits in the binary representation of a. Hence, the binary representation of a^2 is then $(0, a_{m-1}, 0, a_{m-2}, \ldots, 0, a_1, 0, a_0)$. To speed up this computation, it is possible to precompute tables which help to expand the polynomials. An explicit algorithm for polynomial squaring that also includes the precomputation table is given in [HMV04, Algorithm 2.39]. As with the multiplication, the result of a squaring operation is reduced modulo f to obtain a result of degree less than m.

The *inverse* of an element $a \in \mathbb{F}_{2^m}$, denoted by $a^{-1} \mod f$, is the unique element $g \in \mathbb{F}_{2^m}$ such that $ag \equiv 1 \bmod f$. There are various efficient methods to compute the inverse of a field element in $\mathbb{F}_{2^m}$, e.g., the *Euclidean algorithm for polynomials* or the Itoh-Tsujii inversion algorithm [IT88].
We will first show the idea of the Euclidean algorithm by using integers. Then, we will briefly discuss it for polynomials.

Euclidean Algorithm. The authors of [HMV04] describe the *Euclidean algorithm* and its modifications as follows. The classical Euclidean algorithm is used for computing the *greatest common divisor* of two positive integers a and b. The set of integers $\mathbb{Z}$ forms an *Euclidean ring.* In such a ring, we are able to perform a division with remainder, that is, given two values $a, b \in \mathbb{Z}$ with $a \leq b$, we can compute $b = qa + r$ with $q, r \in Z$ and $0 \leq r < a$. Since $\gcd(a, b) = \gcd(b - ta, a)$ for all $t \in \mathbb{Z}$, the computation is reduced to finding $\gcd(r, a)$, where obviously the arguments (r, a) are smaller than (a, b). This computation is repeated until one of the arguments is 0. The result is then directly obtained since $\gcd(0, d) = d$. Due to the fact that the non-negative remainders strictly decrease, this algorithm always terminates.

Extended Euclidean Algorithm. The classical Euclidean algorithm can be extended to additionally obtain the integers x and y satisfying $ax + by = d$ where $d = \gcd(a, b)$. This algorithm, shown in Algorithm 1, terminates as soon as $u = 0$, and hence, $v = \gcd(a, b)$ and $x = x_2$, $y = y_2$ such that $ax + by = d$.

The extended Euclidean algorithm can be used to compute the multiplicative inverse of an element $a \in \mathbb{F}_p$, i.e., $a^{-1} \bmod p$ where p is prime and $a \in \mathbb{F}_p$, as it is detailed in [HMV04, Section 2.2.5].

Extended Euclidean Algorithm for Polynomials. The classical Euclidean algorithm as well as its extended version can be applied to polynomials, based on the fact that $\gcd(a, b) = \gcd(b - ta, a)$ for all binary polynomials t. With the *extended Euclidean algorithm for polynomials*, it is possible to find two binary polynomials g and h, such that $ag + bh = d$ where $d = \gcd(a, b)$. A detailed explanation and an explicit algorithm can be found in [HMV04, Section 2.3.6].

Algorithm 1 Extended Euclidean algorithm [HMV04, Algorithm 2.19]

```
Input: Positive integers $a, b$ with $a \leq b$.
Output: $d = \gcd(a, b)$, $x, y \in \mathbb{Z}$ satisfying $ax + by = d$.
1: $u = a$, $v = b$
2: $x_1 = 1$, $y_1 = 0$, $x_2 = 0$, $y_2 = 1$
3: while $u \neq 0$ do
4:     $q = \lfloor v/u \rfloor$, $r = v - qu$, $x = x_2 - qx_1$, $y = y_2 - qy_1$
5:     $v = u, u = r$, $x_2 = x_1$, $x_1 = x, y_2 = y_1$, $y_1 = y$
6: end while
7: $d = v$, $x = x_2$, $y = y_2$
8: return $(d, x, y)$
```

2.3 Elliptic Curves

An *elliptic curve* E over an arbitrary field K is a *plane*, *smooth* curve defined by the following affine version of the so-called long *Weierstrass equation*:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.3}$$

with coefficients $a_1, a_2, a_3, a_4, a_6 \in K$. The set of all points $(x, y) \in K^2$ satisfying Equation (2.3) plus the point at infinity $\mathcal{O}$:

$$E(K) = \{(x, y) \in K \times K : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\mathcal{O}\}. \tag{2.4}$$

forms an additive Abelian group, where $\mathcal{O}$ is the identity element. The number of points in $E(K)$ is called the *order* of E over K and is denoted by $\#E(K)$.

2.3.1 Discriminant and j-Invariant

An elliptic curve is said to be *smooth* or *non-singular* if there are no points on E at which the curve has more than one distinct tangent line. This condition can be ensured by verifying that the *discriminant* of E, denoted by $\Delta(E)$, is nonzero. Given a curve defined by Equation (2.3), the discriminant $\Delta(E)$ is defined as follows:

$$\left.\begin{aligned}
\Delta(E) &= -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6 \\
b_2 &= a_1^2 + 4a_2 \\
b_4 &= 2a_4 + a_1a_3 \\
b_6 &= a_3^2 + 4a_6 \\
b_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2
\end{aligned}\right\} \tag{2.5}$$

The *j-invariant* of a curve E for $\Delta(E) \neq 0$, denoted by $j(E)$, is defined as

$$j(E) = \frac{b_2^2 - 24b_4}{\Delta(E)}.$$

2.3.2 Group Law

In order to perform computations in the set $E(K)$, an arithmetic operation $+$ is required. An important property of E is that for two distinct points $P, Q \in E(K)$, there is a line

L through P and Q that intersects the curve at a unique third point $R \in E(K)$. In case of $Q = P$, P is counted twice and the line L is the tangent to E at the point P. This tangent again intersects E at a unique third point. This is the basis to define an addition law.

The addition law can be illustrated geometrically with the *chord method* and the *tangent method*.

Addition

The *chord method*, depicted in Figure 2.1, applies if $P \neq Q$. To compute the sum of P and Q, a line L is laid through the two points. L intersects the curve E at a third point $R \in E(K)$. The sum $P + Q$ is actually obtained by reflecting R, i.e. $P + Q = -R$.

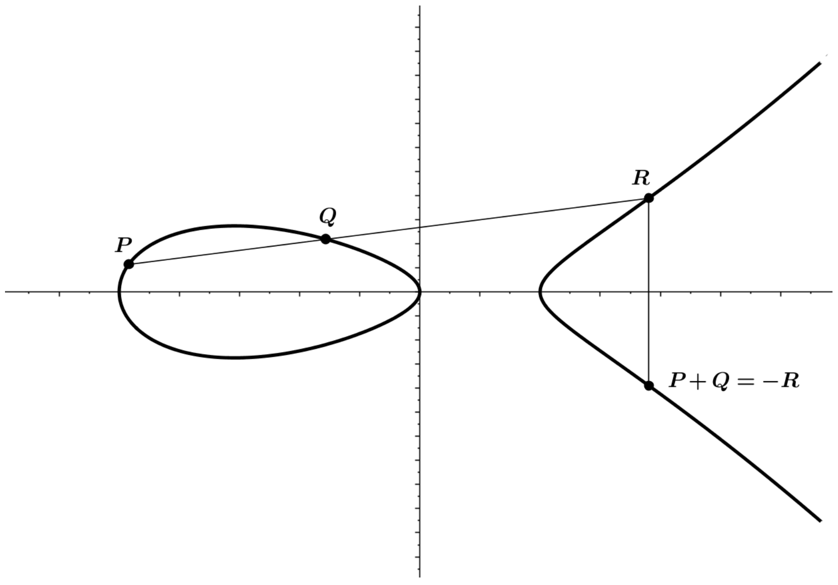


Figure 2.1: Addition of points P and Q on the curve $y^2 = x^3 + 3x^2 - 10x$ over $\mathbb{R}$

Doubling

The *tangent method*, depicted in Figure 2.2, illustrates how a point $P \in E(K)$ is doubled geometrically. In order to compute $P + P = 2P$ a tangent L is laid on E at the point P. L intersects the curve at a third point $R \in E(K)$. If not, the point R is set to $\mathcal{O}$. The point $2P$ is then obtained by reflecting R, i.e, $2P = -R$.

From the geometric descriptions of the point addition and the point doubling, the explicit algebraic formulas can be derived. Let E denote a curve defined by Equation (2.3) and $P = (x_1, y_1)$, $Q = (x_2, y_2) \in E$ be points on the curve. The reflection of point P is

$$-P = (x_1, -y_1 - a_1x_1 - a_3). \tag{2.6}$$

If $Q = -P$ then $P + Q = \mathcal{O}$. Otherwise, if $x_1 \neq x_2$ (*addition*), then

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1},$$
$$\mu = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}.$$

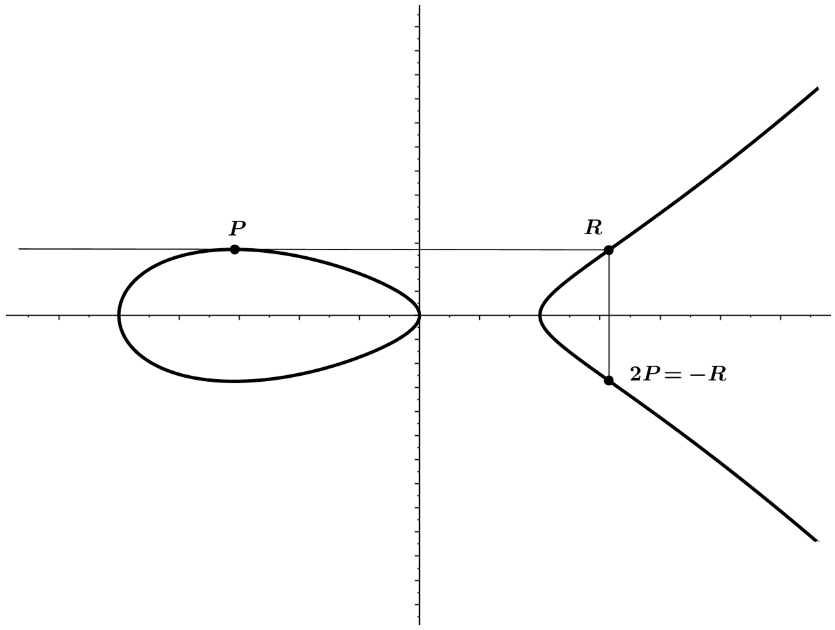


Figure 2.2: Doubling of point P on the curve $y^2 = x^3 + 3x^2 - 10x$ over $\mathbb{R}$

If $x_1 = x_2$ (*doubling*), then

$$\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3},$$
$$\mu = \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x_1 + a_3}.$$

The addition formula for $R = (x_3, y_3) = P + Q \neq \mathcal{O}$ is given by

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2,$$
$$y_3 = -(\lambda + a_1)x_3 - \mu - a_3. \tag{2.7}$$

As already stated, the addition laws together with the corresponding set of points $E(K)$ form an Abelian group $(E(K), +)$ with identity element $\mathcal{O}$. The following group laws are properties of the addition:

- $E(K)$ is closed under the operation $+$, i.e., if $P, Q \in E(K)$ then also $P + Q \in E(K)$.
- The associative law holds in $E(K)$. For all points $P, Q, R \in E(K)$, it applies that $(P + Q) + R = P + (Q + R)$.
- A neutral element $\mathcal{O}$ exists such that $P + \mathcal{O} = \mathcal{O} + P = P, \forall P \in E(K)$.
- For every $P \in E(K)$ there exists an inverse $-P \in E(K)$, such that $(-P) + P = P + (-P) = \mathcal{O}$.
- The operation $+$ is commutative, i.e., for all points $P, Q \in E(K)$ it applies that $P + Q = Q + P$.

2.3.3 Simplified Weierstrass Equations

For fields of specific characteristics, the long Weierstrass equation can be transformed to a simpler form. Such a form is called a *short Weierstrass* form. As a matter of course, the transformation must not change the curve's properties.

Now, consider two elliptic curves E, E' over a field K defined by the following long Weierstrass equations

$$\begin{aligned} E &: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \\ E' &: y^2 + a_1'xy + a_3'y = x^3 + a_2'x^2 + a_4'x + a_6'. \end{aligned}$$

These two curves are *isomorphic over* K if E' can be obtained from E by applying the following change of variables

$$\Psi : (x, y) = (u^2x + r,\ u^3y + u^2sx + t) \tag{2.8}$$

where $u, r, s, t \in K$, $u \neq 0$. The transformation Ψ in Equation (2.8) is referred to as *admissible change of variables*. This admissible change of variables depends on the underlying field K. In this work, we consider the cases $char(K) = 2$ and $char(K) \neq \{2, 3\}$. For other characteristics of K, we refer the reader to [HMV04] and [Eng99].

1. For a field K of characteristic not equal to 2 or 3, the change of variables

$$\Psi : (x, y) = \left(\frac{x - 3a_1^2 - 12a_2}{36},\ \frac{y - 3a_1x}{216} - \frac{a_1^3 - 4a_1a_2 - 12a_3}{24} \right)$$

transforms Equation (2.3) to the short Weierstrass form

$$E : y^2 = x^3 + a_4x + a_6 \tag{2.9}$$

where $a_4, a_6 \in K$, with $\Delta(E) = 16(4a_4^3 + 27a_6^2)$ and $j(E) = 1728 \cdot 4a_4^3 \cdot (4a_4^3 + 27a_6^2)^{-1}$. The addition formulas given in Section 2.3.2 simplify as follows. The reflection given in Equation (2.6) transforms to

$$-P = (x_1,\ -y_1).$$

The addition formula in Equation (2.7) simplifies to

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= (x_1 - x_3)\lambda - y_1, \end{aligned}$$

where if $x_1 \neq x_2$ then

$$\lambda = \frac{y_1 - y_2}{x_2 - x_1},$$

and if $x_1 = x_2$ then

$$\lambda = \frac{3x_1^2 + a_4}{2y_1}.$$

2. For a field K of characteristic 2, there are two cases to be taken into account: *supersingular* curves and *non-supersingular* curves. A curve E defined over a field of characteristic 2 is supersingular if $j(E) = 0$, and non-supersingular otherwise.

(a) If E is *non-supersingular* and, hence, $a_1 \neq 0$, then the change of variables

$$\Psi : (x, y) = \left(a_1^2 x + \frac{a_3}{a_1},\ a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3} \right)$$

transforms Equation (2.3) to the short Weierstrass form

$$E : y^2 + xy = x^3 + a_2 x^2 + a_6 \tag{2.10}$$

where $a_2, a_6 \in K$, with $\Delta(E) = a_6$ and $j(E) = a_6^{-1}$.
The addition formulas given in Section 2.3.2 simplify as follows. The reflection given in Equation (2.6) transforms to

$$-P = (x1,\ y_1 + x_1).$$

The addition formula in Equation (2.7) simplifies to

$$x_3 = \lambda^2 + \lambda + a_2 + x_1 + x_2,$$
$$y_3 = (\lambda + 1)x_3 + \mu,$$

where if $x_1 \neq x_2$ then

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2},$$
$$\mu = \frac{y_1 x_2 + y_2 x_1}{x_1 + x_2},$$

and if $x_1 = x_2$ then

$$\lambda = \frac{x_1^2 + y_1}{x_1},$$
$$\mu = x_1^2.$$

(b) In the case that E is a *supersingular* curve and therefore $a_1 = 0$, then the admissible change of variables

$$\Psi : (x, y) = (x + a_2,\ y)$$

transforms Equation (2.3) to the short Weierstrass form

$$E : y^2 + a_3 y = x^3 + a_4 x + a_6$$

where $a_3, a_4, a_6 \in K$, with $\Delta(E) = a_3^4$ and $j(E) = 0$.
The addition formulas given in Section 2.3.2 simplify as follows. The reflection given in Equation (2.6) transforms to

$$-P = (x_1,\ y_1 + a_3).$$

The addition formula in Equation (2.7) simplifies to

$$x_3 = \lambda^2 + \mu,$$
$$y_3 = \lambda(x_1 + x_3) + y_1 + a_3,$$

where if $x_1 \neq x_2$ then

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2},$$
$$\mu = x_1 + x_2,$$

and if $x_1 = x_2$ then

$$\lambda = \frac{x_1^2 + a_4}{a_3},$$
$$\mu = 0.$$

Throughout the rest of this work we are going to primarily use the short Weierstrass form when it comes to elliptic curve equations.

2.3.4 Projective Coordinate Types

Elliptic curves can be represented in different coordinate systems. In the following we are going to have a look at *projective* coordinate types which have computational advantages over the affine form. As we have seen, the formulas for point addition and point doubling in affine coordinates require field inversions which is the most expensive operation in finite fields. If a field multiplication is considerably cheaper than such a field inversion, it is advisable to use a projective coordinate type for point representation.

Projective Coordinates

Let K be an arbitrary field. The projective equivalent of the affine curve over K, defined by Equation (2.3), is given by

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3. \tag{2.11}$$

The *projective* space $\mathbb{P}^2(K)$ over K is defined as the set of all triples (X, Y, Z) where $X, Y, Z \in K$ and $(X, Y, Z) \neq \mathcal{O}$. These triples have an equivalence relation

$$(X, Y, Z) \sim (X', Y', Z')$$

if there is a $\lambda \in K$ such that

$$X = \lambda^c X', \ Y = \lambda^d Y', \ Z = \lambda Z' \ \text{ with } c, d \in \mathbb{N}.$$

A projective point is denoted by $(X : Y : Z)$. The triple (X, Y, Z) is called a *representative* of $(X : Y : Z)$. Since it applies that if $(X', Y', Z') \in (X : Y : Z)$ then $(X', Y', Z') = (X : Y : Z)$, any element of the equivalence class

$$(X : Y : Z) = \{(\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in K\}$$

can serve as its representative.

If $Z \neq 0$, then $(X/Z^c, Y/Z^d, 1)$ is the affine representative of the projective point $(X : Y : Z)$. There exists a 1-1 correspondence between the set of projective points

$$\mathbb{P}^2(K) = \{(X : Y : Z) : X, Y, Z \in K, Z \neq 0\}$$

and the set of affine points

$$\mathbb{A}^2(K) = \{(x, y) : x, y \in K\}.$$

The set of projective points which do not correspond to any affine point is called *line at infinity.* This line at infinity is the set of projective points where $Z = 0$ and is denoted by

$$\mathbb{P}^2(K)^0 = \{(X : Y : Z) : X, Y, Z \in K, Z = 0\}.$$

The projective version of the non-binary short affine Weierstrass form of $E(K)$, given in Equation (2.9), is obtained by replacing x with X/Z^c and y with Y/Z^d and subsequently clearing the denominators. For the *standard projective coordinates*, where $c = 1$ and $d = 1$, the short Weierstrass form of $E(K)$ is given by

$$E : Y^2Z = X^3 + a_4XZ^2 + a_6Z^3. \tag{2.12}$$

In case of a binary curve, given by Equation (2.10), the projectively closed curve is

$$E : Y^2Z + XYZ = X^3 + a_2X^2Z + a_6Z^3. \tag{2.13}$$

The point at infinity is $\mathcal{O} = (0 : 1 : 0)$, which is the only point of the line at infinity that the curve contains. To map a projective point $(X : Y : Z) \neq \mathcal{O}$ to an affine point, one simply computes $(X/Z^c : Y/Z^d)$. To map an affine point $(X : Y)$ which is not at infinity, one chooses $Z = 1$ and computes $(XZ^c : YZ^d : Z)$.

Projective Point Addition Formulas

The following equations will give the addition formula and doubling formula with the best operation counts according to [BL14] for standard projective coordinates.

Addition. Formula for computing $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ on a curve defined by Equation (2.13):

$$\begin{aligned}
A &= Y_1Z_2 + Z_1Y_2 & B &= X_1Z_2 + Z_1X_2, & C &= B^2,\\
D &= Z_1Z_2, & E &= BC, & F &= A + B,\\
G &= (AF + a_2C)D + E, & & & &\\
X_3 &= BG, & Y_3 &= C(AX_1Z_2 + BY_1Z_2) + FG, & Z_3 &= ED.
\end{aligned}$$

Doubling. Formula for computing $2(X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ on a curve defined by Equation (2.13):

$$\begin{aligned}
A &= X_1^2, & B &= A + Y_1Z_1, & C &= X_1Z_1,\\
D &= C^2, & E &= B + C, & F &= B * E + a_2D,\\
X_3 &= CF, & Y_3 &= EF + A^2C, & Z_3 &= CD.
\end{aligned}$$

As we can see, in projective coordinates no inversion operation is needed. Using the formulas above, an addition requires $\texttt{14M} + \texttt{1S}$ and a doubling can be done with $\texttt{7M} + \texttt{3S}$, where $\texttt{M}$ denotes the cost of one field multiplication and $\texttt{S}$ denotes the cost of one field squaring. Additional formulas for standard projective coordinates can be found in [BL14] and [CFA⁺05, Section 13.2.1.b].

Jacobian Coordinates

Jacobian Coordinates are projective coordinates where $c = 2$ and $d = 3$. They offer fast doubling formulas. The negative of the projective point $P = (X : Y : Z)$ is $-P = (X : X + Y : Z)$ and P represents the affine point $\bar{P} = (X/Z^2, Y/Z^3)$. The point at infinity $\mathcal{O}$ is defined as $(1 : 1 : 0)$. Thus, using Jacobian coordinates, Equation (2.9) evolves to

$$E : Y^2 = X^3 + a_4XZ^4 + a_6Z^6$$

and Equation (2.10) evolves to

$$E : Y^2 + XYZ = X^3 + a_2X^2Z^2 + a_6Z^6.$$

There are formulas for Jacobian coordinates so that a point addition can be done in $\mathtt{14M} + \mathtt{5S}$ and a doubling operation in $\mathtt{4M} + \mathtt{5S}$. For the explicit formulas, we refer the reader to [BL14] as well as [CFA+05, Section 13.3.1.c].

López-Dahab Coordinates

In 1998, López and Dahab introduced a projective coordinate type for binary curves, where $c = 1$ and $d = 2$ [LD98]. In this set of coordinates, the projective equation of the curve defined by the Equation (2.10) is given as

$$E : Y^2 + XYZ = X^3Z + a_2X^2Z^2 + a_6Z^4.$$

The negative of the projective point $P = (X : Y : Z)$ with $Z \neq 0$ is $-P = (X : X + Y : Z)$ and P represents the affine point $\bar{P} = (X/Z, Y/Z^2)$. The point at infinity $\mathcal{O}$ is defined as $(1 : 0 : 0)$. With López-Dahab coordinates, the point addition can be done in $\mathtt{13M} + \mathtt{4S}$ and a doubling requires $\mathtt{3M} + \mathtt{5S}$. Explicit formulas for these coordinates can be found in [BL14] as well as [CFA+05, Section 13.3.1.d].

2.4 Summary

This chapter gave an overview of the mathematical principles that are necessary to understand the remainder of this master's thesis. It discussed the algebraic concepts of groups, fields and rings and introduced the discrete logarithm problem as well as the Freshman's dream and Lucas sequences. Furthermore, the chapter provided some background on binary fields and the corresponding arithmetic. On the basis of these concepts, the last part of this chapter explained the fundamentals of elliptic curves, like the group law and Weierstrass equations, and detailed projective coordinate types for point representation on elliptic curves.

Chapter 3

Security Properties of Elliptic Curves

This chapter addresses the security properties of elliptic curves over finite fields. It details the elliptic curve discrete logarithm problem (ECDLP) and a selection of attacks on it. The presented attacks are divided into two categories. At first, there are generic attacks, which do not exploit any special structure of the curve. Then, there are specific attacks that make use of the curve's structure, e.g. by exploiting an isomorphism available on the curve. The chapter consists of five sections. Section 3.1 and Section 3.2 present the elliptic curve logarithm problem (ECDLP) and the elliptic curve Diffie-Hellman problem (ECDHP), respectively. Subsequently, Section 3.3 is dedicated to generic attacks on the ECDLP and details the Pollard's rho attack. Curve-specific types of attacks are presented in Section 3.4. This section covers the SSSA attack, an isomorphism attack suitable for prime-field anomalous curves, as well as the Weil pairing attack or MOV attack, which is an attack feasible on supersingular elliptic curves. Finally, the chapter is briefly summarized in Section 3.5.

3.1 Elliptic Curve Discrete Logarithm Problem

In Section 2.1.1, we have introduced the discrete logarithm problem (DLP). Now, we are going to have a look at its counterpart on elliptic curves.

Definition 3.1 (ECDLP)**.** Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ and let $G \in E(\mathbb{F}_q)$ be a point of order n. Given a point $P \in \langle G \rangle$, the *elliptic curve discrete logarithm problem* (ECDLP) is the problem of finding an integer $0 \leq k < n$ so that $P = kG$.

The hardness of the ECDLP is the basis that all security assumptions of elliptic curves rely on. Hence, it is essential to carefully choose the elliptic curve parameters in cryptographic schemes in order to resist all known attacks on the ECDLP.

The most obvious and at the same time most impracticable attack is certainly the exhaustive search where simply the sequence $G, 2G, 3G, \ldots$ is computed until P is found. This method has an expected running time of $n/2$ and a running time of n in the worst case. In the Sections 3.3 and Section 3.4, we are going to present a selection of far more sophisticated attacks on the ECDLP.

3.2 Elliptic Curve Diffie-Hellman Problem

The elliptic curve Diffie-Hellman problem (ECDHP) is the elliptic curve analogue to the Diffie-Hellman problem (DHP) discussed in Section 2.1.1.

Definition 3.2 (ECDHP)**.** Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ and $G \in E(\mathbb{F}_q)$ be a point of order n. Given two points $P = kG, Q = lG \in \langle G \rangle$, the problem of finding a point $R = klG$ is called the elliptic curve Diffie-Hellman problem (ECDHP).

The ECDHP is not harder than the ECDLP since, if the ECDLP can be efficiently solved for $P = kG$, then it is easy to solve the ECDHP subsequently by computing $R = kQ$.

3.3 Generic Attacks

In this section we are going to detail a generic attack on the ECDLP. As the authors of [CFA$^+$05] state, an attack is *generic*, if it only performs computations of the composition of two elements, computations of the inverse of an element, and comparisons of two elements. In other words, generic attacks are attacks that can be used in any DLP settings because they do not involve any special structure of the elliptic curve.

3.3.1 Pollard's Rho

In 1978, J. Pollard proposed an algorithm for solving the DLP in [Pol78]. The proposed algorithm is an analogue to the integer factorization algorithm he proposed earlier in [Pol75]. The *Pollard's rho for logarithms* algorithm searches for collisions in pseudo-random sequences, very similar to Floyd's cycle-finding algorithm. A big advantage of this algorithm is the negligible amount of required storage. However, it is a Monte Carlo algorithm, which means that there is no guarantee of success. In the following we are going to explain how Pollard's rho algorithm is used to solve the ECDLP. The section is based on the details given in [HMV04] and [Sma12], as well as [KS01] and [YISK11].

Let $G \in E(\mathbb{F}_q)$ be a base point of order n and $P \in \langle G \rangle$, then, in order to solve the ECDLP, we have to find a $k \in [0, n-1]$ so that $P = kG$. The goal of the algorithm is to find two distinct pairs (c, d) and $(\bar{c}, \bar{d})$ of integers modulo n where

$$cG + dP = \bar{c}G + \bar{d}P. \tag{3.1}$$

As $P = kG$, Equation 3.1 can be written as follows:

$$cG + dkG = \bar{c}G + \bar{d}kG. \tag{3.2}$$

In terms of modular arithmetic with respect to the group order n, we get

$$c + dk \equiv \bar{c} + \bar{d}k \bmod n, \tag{3.3}$$

and to obtain k we compute

$$k = \log_G P = (c - \bar{c})(\bar{d} - d)^{-1} \bmod n. \tag{3.4}$$

A simple approach is to repeatedly generate a random pair (c, d) and store triples $(c, d, cG + dP)$ ordered by the third entry for each generated pair. This is done until a point $cG + dP$ occurs a second time. If the corresponding value d of these two points is not the same, then a collision has been found. The expected number of required iterations is according to the birthday paradox $\sqrt{\frac{\pi n}{2}} = 1,2533\sqrt{n}$. The big disadvantage of this approach is the required storage, which is expected to be $\sqrt{\frac{\pi n}{2}}$ triples, since every triple has to be stored.

The approach proposed by Pollard has roughly the same expected running time but is in fact *memoryless*. To do so, it is necessary to define an *iterating function* $f : \langle G \rangle \to \langle G \rangle$ that is fast to compute. In addition, f should have random-function characteristics. Typically, $f(X) = \bar{X} = \bar{c}G + \bar{d}P$ with $\bar{c}, \bar{d} \in [0, n-1]$ is chosen as iterating function.

The iterating function f is used to define a sequence $\{X_i\}$ by $X_{i+1} = f(X_i)$ where $i \geq 0$, with a random starting value X_0. Due to the random-function characteristics of f, this sequence behaves as a *pseudorandom walk* in $\langle G \rangle$.

Consequently, any point X_0 determines a sequence $\{X_i\}_{i \geq 0}$. Because the set $\langle G \rangle$ is finite, the sequence will collide at some point and from there on it will cycle forever. This happens at the point where $X_t = X_{t+s}$ for some $s \geq 1$ and index t is minimal. From this point on $X_i = X_{i-s}$ for all $i \geq t + s$, which means that the sequence cycles. Figure 3.1 graphically illustrates such a cycling sequence. Due to its shape, which bears a strong resemblance to the Greek letter ρ, the algorithm is called *Pollard's rho* algorithm.

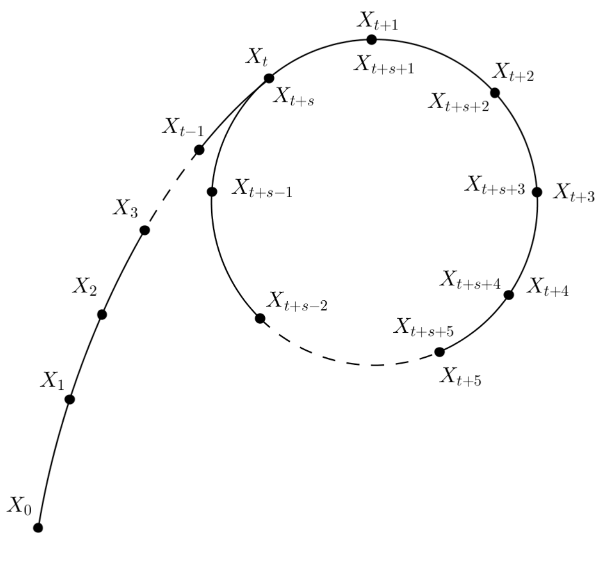


Figure 3.1: Resulting sequence $\{X_i\}$ in Pollard's rho algorithm

The index t is called the *tail length* and s is called the *cycle length*. As the iterating function f has a random characteristic, the expected number of iterations before obtaining a collision is approximately $\sqrt{\frac{\pi n}{2}}$. The expected length of t and s is approximately $\sqrt{\frac{\pi n}{8}}$ each. The complete algorithm, based on information from [HMV04], is detailed in Algorithm 2. To find a collision of two points X_i, X_j where $X_j = X_j$ with $i \neq j$, the well-known Floyd's cycle finding algorithm ([Knu69], [Flo67]) can be used.

Algorithm 2 Pollard's rho algorithm

```
Input: $G \in E(\mathbb{F}_q)$ of order $n$, where $n$ is prime, $P \in \langle G \rangle$.
Output: $\log_G(P)$.
 1: Select the number of branches $L$.
 2: Select a partition function $H : G \to \{1, 2, \ldots, L\}$
 3: for $j = 1$ to $L$ do
 4:     Select $a_j, b_j \in_R [0, n-1]$
 5:     $Q_j = a_j + b_j Q$
 6: end for
 7: Select $c, d \in_R [0, n-1]$
 8: $X = cG + dP$
 9: $\bar{X} = X$, $\bar{c} = c$, $\bar{d} = d$
10: do
11:     $j = H(X)$
12:     $X = X + Q_j$, $c = c + a_j \mod n$, $d = d + b_j \mod n$
13:     for i = 1 to 2 do
14:         $j = H(\bar{X})$
15:         $\bar{X} = \bar{X} + Q_j$, $\bar{x} = \bar{c} + a_j \mod n$, $\bar{d} = \bar{d} + b_j \mod n$
16:     end for
17: while $X \neq \bar{X}$
18: if $d = \bar{d}$ then
19:     return (failure)
20: else
21:     $l = (c - \bar{c})(\bar{d} - d)^{-1} \mod n$
22:     return $l$
23: end if
```

Iterating Function

An example for an *iterating function* is the *L-adding walk* proposed by Teske in [Tes00]. The set $\langle G \rangle$ is quasi randomly partitioned into L sets $\{S_1, S_2, S_3, \ldots, S_L\}$ of roughly the same size. L denotes the number of branches. A point $X \in \langle G \rangle$ can be assigned to a set S_j depending on the number of branches. For example, let $L = 2^z$, then X is assigned to the set S_j if the value represented by the z least significant bits of the x-coordinate of X is equal to the integer $j - 1$. Typical values for L are 16 and 32. If $X \in S_j$ then this is written as $H(X) = j$ where H is called the *partition function*. With $a_j, b_j \in_R [0, n-1]$ for $1 \leq j \leq L$, the iterating function $f : \langle G \rangle \to \langle G \rangle$ is defined by:

$$f(X) = X + a_j P + b_j Q \text{ where } j = H(X). \tag{3.5}$$

If $X = cG + dP$ is given, we can compute $f(X) = \bar{X} = \bar{c}G + \bar{d}P$ where $\bar{c} = c + a_j \mod n$ and $\bar{d} = d + b_j \mod n$.

Parallelization

The Pollard's rho algorithm detailed above uses only a single processor, however, it is possible to extend the number of employed processors. In [OW99], Van Oorschot and Wiener proposed a parallelized variant of Pollard's rho that can employ an arbitrary number of processors, where for M used processors it yields a speedup of factor M. Therefore a *distinguishing property* for points is selected, i.e., a property that can easily be tested, such as the number of leading zeros of a point's x-coordinate. Each processor starts with its own randomly selected starting point X_0, but all use the same iterating function f to generate the sequences $\{Xi\}i_{\geq 0}$. If a sequence hits a distinguished point, this point is sent to a central server which stores all those points in a sorted list. As soon as the server receives a stored point a second time, the discrete logarithm can be computed via Equation (3.4). The expected running time of the parallelized Pollard's rho algorithm is $\sqrt{(\pi n/2)}/M + 1/\theta$, where θ denotes the proportion of points in $\langle P \rangle$ that have the selected distinguishing property. For a more detailed explanation and an explicit algorithm, we refer the reader to [HMV04, Section 4.1.2].

3.4 Specific Attacks

This section is dedicated to attacks on the ECDLP that make use of the curve's structure. In the following, we are going to focus solely on isomorphism attacks. These attacks reduce the effort required to solve the ECDLP by using an isomorphism provided by the elliptic curve.

The principle of *isomorphism attacks* is to map elements of elliptic curve groups to isomorphic groups, where the DLP is less hard, i.e., where the DLP can be solved in subexponential or even polynomial time. The solution to the DLP in the other group then immediately gives a solution to the DLP in the elliptic curve group. Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ and $G \in E(\mathbb{F}_q)$ a point of prime order n. Moreover, let $\mathcal{G}_\mathcal{T}$ be a group of order n as well. Then, $\langle G \rangle$ and $\mathcal{G}_\mathcal{T}$ are isomorphic since both groups have order n. Whenever an isomorphism

$$\phi : \langle G \rangle \to \mathcal{G}_\mathcal{T}$$

can be computed efficiently, the ECDLP instances in $\langle G \rangle$ can be transferred to DLP instances in $\mathcal{G}_\mathcal{T}$.

These attacks are *specific* or *special purpose* as they can solve the ECDLP faster than Pollard's rho only on specific classes of elliptic curves. Such specific classes are, for example, the class of *prime-field anomalous curves* and the class of *supersingular curves.*

3.4.1 Isomorphism Attacks on Prime-Field Anomalous Curves

An elliptic curve E defined over a prime field $\mathbb{F}_p$, where $\#E(\mathbb{F}_p) = p$, is called a *prime-field anomalous* curve. The group $E(\mathbb{F}_p)$ has prime order and is thus cyclic. As a consequence $E(\mathbb{F}_p)$ is isomorphic to the additive group $(\mathbb{Z}_p, +)$ of integers modulo p. In this additive group, the DLP is defined as the problem of finding $l \in [0, p-1]$ such that

$$la \equiv b \pmod{p} \tag{3.6}$$

with $p, a, b \in (\mathbb{Z}_p, +)$ and $a \neq 0$.

From Equation (3.6) we can deduce that $l = ba^{-1} \mod p$, and, thus, we can use the extended Euclidean algorithm (see Section 2.2.1) to compute $a^{-1} \mod p$ and, hence, solve the DLP in $(\mathbb{Z}_p, +)$ efficiently.

The SSSA Attack

In 1997, Semaev [Sem98], Smart [Sma99] and Satoh and Araki [SA98] independently showed how to efficiently compute an isomorphism

$$\phi : E(\mathbb{F}_p) \to (\mathbb{Z}_p, +)$$

for prime-field anomalous elliptic curves. Hence, this so-called SSSA attack, illustrated in Algorithm 3, gives a polynomial-time algorithm for the ECDLP in $E(\mathbb{F}_p)$. However, it solely applies to prime-field anomalous curves and cannot be extended to other classes of elliptic curves. The SSSA attack can easily be circumvented by avoiding curves whose number of points is equal to the cardinality of the underlying field, i.e, by avoiding curves where $\#E(\mathbb{F}_p) = p$.

Algorithm 3 SSSA algorithm

```
Input: $G \in E(\mathbb{F}_p)$, $P \in \langle G \rangle$ and $\phi(P) \neq 0$.
Output: Integer $l$ s.t. $P = lG$.
1: $a = \phi(P)$, $b = \phi(G)$
2: Find $l \in [0, p-1]$ : $l \equiv ba^{-1} \bmod p$        ▷ using the extended Euclidean algorithm
3: return $l$
```

3.4.2 The MOV Attack

In the following, we are going to detail the Weil pairing attack. This part is based on the works [Sil09], [Mil04] and [Aft11]. In 1991, Menezes, Okamoto and Vanestone showed an approach to attack supersingular curves using the Weil pairing [MOV91]. The so-called MOV attack reduces the ECDLP in a curve $E(\mathbb{F}_q)$ to the DLP in an extension field $\mathbb{F}_{q^k}$ of $\mathbb{F}_q$. The reduction is using an isomorphism between the subgroup of E generated by G, where G is of order n, denoted by $\langle G \rangle$ and the subgroup of n-th roots of unity in the extension field $\mathbb{F}_{q^k}$. Here, the integer k, which is called the *embedding degree*, is the smallest positive integer such that $n|(q^k - 1)$.

Here, we present a simplified version of the MOV attack, which can be achieved for any elliptic curve equipped with a symmetric pairing, such that the embedding degree is small, i.e., $k \leq 6$. This applies solely to supersingular curves. Note that for higher embedding degrees, the reduction of the ECDLP to the DLP in the subgroup $\mathcal{G}_\mathcal{T}$ of $\mathbb{F}_{q^k}$ gives no advantage over solving the ECDLP directly.

The Pairing

Let $\langle G \rangle$ denote an elliptic curve subgroup of prime order p of $E(\mathbb{F}_q)$, generated by G, and $\mathcal{G}_\mathcal{T}$ a multiplicative subgroup of the finite field $\mathbb{F}_{q^k}$ of prime order p. A symmetric bilinear pairing is then a map of the form:

$$e : \langle G \rangle \times \langle G \rangle \to \mathcal{G}_\mathcal{T}. \tag{3.7}$$

A bilinear pairing has the following important properties:

1. *Bilinearity*: $\forall P, Q, R \in \langle G \rangle$:

$$e(P + Q, R) = e(P, R) \cdot e(Q, R)$$
$$e(P, Q + R) = e(P, Q) \cdot e(P, R)$$

2. *Non-degeneracy*: $\forall P \in \langle G \rangle$: if $P \neq \mathcal{O}$ then $e(P, P)$ is a generator of $\mathcal{G}_\mathcal{T}$, i.e.,
$$e(P, P) \neq 1.$$
3. *Computability*: $e(P, Q)$ is efficiently computable for all $P, Q \in \langle G \rangle$.

The Algorithm

Algorithm 4 states the simplified version of the MOV attack using the symmetric bilinear pairing e defined in Equation (3.7).

Algorithm 4 Simplified MOV algorithm

```
Input: $G \in E(\mathbb{F}_q)$, $ord(G) = p$ where $p$ is prime, $P \in \langle G \rangle$.
Output: Integer $l$, s.t. $P = lG$.
1: $\alpha = e(G, G) \neq 1$
2: $\beta = e(G, P)$
3: Use subexponential algorithm to compute the discrete logarithm of $\beta$ to the base $\alpha$,
   i.e., find $l \in \mathbb{N}$, s.t. $\beta = \alpha^l$
4: return $l$
```

The reduction of the ECDLP in $E(\mathbb{F}_q)$ to the DLP in the extension field $\mathbb{F}_{q^k}$ of $\mathbb{F}_q$ is feasible due to the bilinearity as well as the non-degeneracy of e. We know from the non-degeneracy that $e(G, G)$ is a generator element of $\mathcal{G}_\mathcal{T}$, denoted by g in the following. We also know due to the bilinearity of e that $e(P + Q, R) = e(P, R) \cdot e(Q, R)$. Thus, we have
$$e(G, P) = e(G, lG) \overset{\text{bilinear}}{=} \prod_{i=1}^{l} e(G, G) = e(G, G)^l = g^l$$
Once we have $\alpha = e(G, G)$ and $\beta = e(G, P)$, we can solve $\beta = \alpha^l$ in subexponential time by using *index-calculus* methods.

Index-Calculus

The *index-calculus* is a probabilistic method to compute discrete logarithms in some groups. The method does not apply to every group, but when it does, it is possible that it yields a subexponential-time algorithm. The following explanations are based on the details given in [HMV04] and [MVO96], as well as on [Sah12] and [Die12].

We start by illustrating the main idea of index-calculus on the basis of a cyclic group $\mathcal{G}$. Let $\mathcal{G}$ be of order n, generated by α. First, select a *factor base*. That is a subset $S = \{p_1, p_2, \ldots, p_t\}$ of $\mathcal{G}$ such that a significant fraction of elements of $\mathcal{G}$ can be efficiently expressed as a product of elements from S.

Next, we have to find linear relations which solve logarithms of elements in S. Therefore, we select random integers $k \in [0, n-1]$ until a^k is S*-smooth*, meaning that a^k splits into elements in S. When such a k is found, a^k can be completely factored, i.e.,
$$a^k = p_1^{c_1} \ldots p_t^{c_t}, \text{ where } c_i \geq 0. \tag{3.8}$$
When taking the logarithms to the base α of both sides of Equation (3.8), we obtain a linear equation
$$k \equiv \sum_{i=1}^{t} c_1 \log_\alpha p_1 \pmod{n}, \tag{3.9}$$

which is a “relation of indices”. *Index* is the original name for *discrete logarithm.* In Equation (3.9) the unknowns are the logarithms of the factor base elements (indices).

We repeat the steps above until we get $t + \delta$ relations of the form of Equation (3.9), where δ is a small positive integer, e.g., $\delta = 4$. The additional δ relations are meant to ensure that the resulting system of equations, consisting of $t + \delta$ relations, has a unique solution with high probability, i.e., t of the $t + \delta$ equations are linearly independent.

Now, we have to find the logarithms of elements in S. This is achieved by solving this linear system and, thus, obtaining $\log_\alpha p_i$ for $i \in [1, t]$. Subsequently, we compute $\log_\alpha \beta$. Therefore, we select random integers $k \in [0, n-1]$ until $\alpha^k\beta$ is S-smooth and can be written as linear relation of the form

$$\alpha^k\beta = \prod_{i=1}^{t} p_i^{d_i}, \text{ where } d_i \geq 0. \tag{3.10}$$

The last step, in order to obtain the desired logarithm of β, is taking logarithms to the base α on both sides of Equation (3.10):

$$\log_\alpha \beta = -k + \sum_{i=1}^{t} d_i \log_\alpha p_i \bmod n.$$

This is the basic idea of the index-calculus method.

The running time of this method depends heavily on the choice of the factor base S. As outlined in [HMV04], there is also a trade-off in the size t of S. On the one hand we want to have a larger t because it increases the probability that a random group element factors over S. On the other hand we want a small t because then the number of relations to collect is smaller. Thus, the optimal size t is determined by the proportion of elements in $\mathcal{G}$ that factor over S.

Note that the presented index-calculus method cannot in general be used in a group of points on elliptic curves. The problem is that there is no equivalence to the prime elements in such a group and, hence, it is not possible to find an efficient factor base.

Index-Calculus in $\mathbb{F}_p^*$. The elements of the multiplicative group of a prime field $\mathbb{F}_p^*$ can be regarded as the integers in $[1, p-1]$. The evident choice for the factor base S are the first t prime numbers, where t is at most a prescribed bound B. Every element of $\mathbb{F}_p^*$ whose prime factors are $\leq B$ is B-smooth and, thus, factors over S. In the basic explanation of the index-calculus method above, we mentioned that the optimal size t of the factor base S depends on the proportion of elements that factor over S. In the case of a prime field, the optimal choice of t depends on the distribution of B-smooth integers in $[1, p-1]$. With an optimal choice of t, we can obtain a subexponential-time algorithm for the *DLP* in F_p^*. According to [HMV04], the fastest variant of this algorithm to compute logarithms in $\mathbb{F}_p^*$ is the number field sieve, which has subexponential running time.

Index-Calculus in $\mathbb{F}_{2^m}^*$. The elements of the multiplicative group of a binary field $\mathbb{F}_{2^m}^*$ can be regarded as nonzero polynomials in $\mathbb{Z}_2[x]$ of degree at most $m - 1$. Thus, the evident choice for the factor base S is the set of all irreducible binary polynomials of at most some prescribed bound B. Every element of $\mathbb{F}_{2^m}^*$ for which it applies that all its irreducible factors have a degree of at most B, is B-smooth and, thus, factors over S. The optimal size of S in the case of a binary field depends on the distribution of polynomials that are B-smooth among all the binary polynomials of degree less than m. yields an subexponential-time algorithm for the DLP in $\mathbb{F}_{2^m}^*$. The fastest index-calculus variant for computing logarithms in $\mathbb{F}_{2^m}^*$ is Coppersmith's algorithm and derivatives thereof, as stated in [HMV04] and [MVO96].

In 2013, Antoine Joux proposed a new index-calculus algorithm in [Jou13a], which can be efficiently computed in small characteristic. This algorithm is suitable for all finite fields that are relevant for pairings, as it is the case for the MOV attack. As a consequence, symmetric pairings on binary curves can be considered as broken.

3.5 Summary

This chapter gave insight into the security properties of elliptic curves. It detailed the ECDLP as well as the ECDHP, and presented different attacks on the ECDLP on elliptic curves. Within the stated attacks, we differentiated between two types: Firstly, attacks that work on the ECDLP as well as on the DLP as they do not employ any special structure of the elliptic curve. These attacks were so-called generic attacks. Secondly, we described specific attacks, which make use of a structure provided by the elliptic curve, such as an isomorphism. This type of attacks applies to a certain class of curves only, e.g., supersingular curves. The chapter also gives a rough idea why it is important to choose the elliptic curve's parameters carefully.

Chapter 4

Protocols

This chapter gives an overview of four different cryptographic protocols on elliptic curves. A cryptographic protocol can be seen as a series of computation steps and message exchanges between two or more entities. The goal is to achieve a certain security objective, e.g., a digital signature or a shared secret key, in a secure manner. In this chapter, a signature scheme, two key-agreement protocols, and a hybrid encryption scheme are going to be explained. We are also going to detail the objectives of each protocol and give explicit algorithms thereof.

This chapter is structured as follows. At first, Section 4.1 gives a superficial overview of the domain parameters which are essential for almost all ECC-based protocols. Then, in Section 4.2 the Elliptic Curve Digital Signature Algorithm (ECDSA) is introduced and its signature generation and signature verification is explained, along with a brief glimpse into public-key validation. The next section, Section 4.3, is dedicated to the Elliptic Curve Diffie-Hellman (ECDH) key agreement. It is followed by a discussion of the the ECMQV key agreement in Section 4.4. The last protocol described in this chapter is the Elliptic Curve Integrated Encryption Scheme (ECIES), which is illustrated in Section 4.5. At the end, this chapter is briefly summarized in Section 4.6.

4.1 Domain Parameters

Domain parameters are a set of elements to identify a certain elliptic curve. It is essential that all parties agree on the scheme's domain parameters, since the scheme usually requires that all participating entities perform computations on the same curve. The elements contained in the domain parameters may vary depending on the scheme. The domain parameters D described below are required for all of the following elliptic-curve schemes. They specify an elliptic curve E over a finite field $\mathbb{F}_q$, a base point $G \in E(\mathbb{F}_q)$ and the base point's order n. Additionally, they describe an indication field representation used for the elements in $\mathbb{F}_q$. The domain parameters $D = (q, FR, S, a, b, G, n, h)$ consist of the following values:

- a field order q, where either $q = p$ and p is a prime > 2, or $q = 2^m$,
- a field representation FR of the representation used for the elements of $\mathbb{F}_q$,
- an optional bitstring seed S if the curve was generated randomly,
- two field elements $a, b \in \mathbb{F}_q$ (called *coefficients*) that define the equation of the curve E over $\mathbb{F}_q$ (the equation of E depends on the type of the underlying field),

- a finite point $G = (x_G, y_G) \in E(\mathbb{F}_q)$ of prime order, called the *base point*,
- the order n of the base point G, and
- the cofactor $h = \#E(\mathbb{F}_q)/n$.

If a scheme explicitly uses standardized curves (cf. [Nat13], [Cer00], [LM10]), the domain parameters are sometimes given as $D = (E, G, n)$ for convenience where E is the standardized curve, G is a finite point on the curve E (base point) and n is the order of point G.

4.1.1 Generation and Validation of Domain Parameters

The generation and validation of the domain parameters will not be dealt with in detail in this work, because in practice the domain parameters are not generated each time by (one of) the participants since this procedure is quite time consuming and the implementation itself is quite error-prone. Instead, there are domain parameters for several common field sizes published by various organizations and institutes, like NIST [Nat13] or SECG [Cer00]. These domain parameters are often called *named curves* or *standard curves.* Such a standard curve is either identified by its unique name or by its unique object identifier (OID). Both identifiers are defined in the corresponding standardization document. For a more detailed view on the explicit domain parameter generation and verification we refer the reader to [JMV01] and [HMV04].

4.2 ECDSA

The *Elliptic Curve Digital Signature Algorithm* (ECDSA) is the elliptic curve variant of the Digital Signature Algorithm (DSA). In 1991, NIST proposed DSA which was then specified under the name Digital Signature Standard (DSS) in FIPS 186 [Nat00]. In 1992, Scott Vanstone proposed ECDSA as a response the NIST's request for public comments on their DSS [Van92]. Since 1998, ECDSA has been widely standardized. It appears in the standards ISO/IEC 14888-3 [ISO98], ANSI X9.62 [ANS98], IEEE 1363-2000 [IEE00], and FIPS 186-3 [Nat13]. A very detailed description of this topic can be found in [JMV01] which also served as basis for this section.

4.2.1 Signature Generation

The generation of an ECDSA signature is detailed in Algorithm 5. The input of the ECDSA signature generation algorithm consists of the domain parameters D, a private key d and the message m to create the signature on.

The function $H(m)$ in Step 4 denotes a cryptographic hash function, such as `SHA-1`. The result e of this hash function is required to have a maximum bitlength of n bits. If the output of $H(m)$ is larger, it can be truncated, such that e represents the n leftmost bits of $H(m)$.

Algorithm 5 ECDSA signature generation

```
Input: Domain parameters $D = (q, FR, S, a, b, G, n, h)$, private key $d$, message $m$.
Output: Signature $(r, s)$.
1: Select a random integer $k \in_R [1, n-1]$.
2: $Q = (x_1, y_1) = kG$    ▷ Key generation: $k$ is the ephemeral private key and $Q$ is the public key
3: $r = x_1 \bmod n$, if $r = 0$ go to Step 1.
4: $e = H(m)$    ▷ $H$ denotes a cryptographic hash function, such as SHA-1
5: $s = k^{-1}(e + dr) \bmod n$, if $s = 0$ go to Step 1.
6: return $(r, s)$.
```

It is important that the ephemeral private key k, also referred to as the *per-message secret*, in Step 1 is generated randomly for every signature and discarded after each use. These secrets must be kept private at all times, because an adversary knowing the ephemeral private k that was used to generate a signature (r, s), is to able recover the signer's private key d since

$$d = r^{-1}(ks - e) \bmod n. \tag{4.1}$$

Thus, the per-message secrets k have the same security requirements as the private key d. Furthermore, it is required that per-message secrets never repeat, because this would enable the recovery of k and subsequently the signer's private key d, as illustrated in the following.

Let (r, s_1) and (r, s_2) be two ECDSA signatures, generated with the same private key d and the same k on two different messages m_1 and m_2. Let $e_1 = H(m_1)$ and $e_2 = H(m_2)$. According to Step 5 of Algorithm 5 it applies that $s_1 \equiv k^{-1}(e_1 + dr) \pmod n$ and $s_2 \equiv k^{-1}(e_2 + dr) \pmod n$. Consequently, $ks_1 \equiv e_1 + dr \pmod n$ and $ks_2 \equiv e_2 + dr \pmod n$. By subtracting these two equations, we get $k(s_1 - s_2) \equiv e_1 - e_2 \pmod n$ and, therefore, we obtain

$$k \equiv (s_1 - s_2)^{-1}(e_1 - e_2) \pmod n,$$

if $s_1 \not\equiv s_2 \pmod n$, which is the case with overwhelming probability if H is a secure hash function. Once an adversary has learned k he is able to recover d via Equation (4.1).

4.2.2 Signature Verification

Prior to the actual ECDSA signature verification, it is essential to validate the domain parameters associated with the signature. In the following, we are going to give a short explanation of the validation of the signer's public key.

Public-Key Validation

Public-key validation is essential for almost all elliptic-curve protocols. It was devised the first time by Johnson in [Joh97]. This validation ensures that a public key has the requisite arithmetical properties and should primarily prevent malicious insertions of an invalid public key. In the worst case the use of an invalid public key can void all expected security properties.

The following checks explicitly validate an ECDSA public key Q associated with valid domain parameters:

- Check that the point is not at infinity, i.e, $Q \neq \mathcal{O}$.

- Check that x_Q, y_Q are properly represented elements of the underlying field $\mathbb{F}_q$.
- Check that $Q \in E(\mathbb{F}_q)$.
- Check that $nQ = \mathcal{O}$.

If all these checks are successful, then Q is a valid public key. However, it is neither ensured that the alleged owner of Q is in possession of the associated private key nor that this private key was generated at all.

To verify a signature (r, s) on a message m, the verifier V has to obtain the signature (r, s), the exact domain parameters D the signer used and the signer's associated public key Q, which, as the very first step, has to be validated as described above. In practice, the public key is additionally authenticated, e.g., by an `X.509` certificate.

The verification of an ECDSA signature in detail is depicted in Algorithm 6. Note that the hash function H used in Step 2 is required to be the same that was used to generate the signature. Again, similar to the generation, the result of $H(m)$ might get truncated in order to have bitlength of at most n bits.

Algorithm 6 ECDSA signature verification

```
Input: Domain parameters $D = (q, FR, S, a, b, G, n, h)$, public key $Q$, message $m$, signature $(r, s)$.
Output: Accept or reject the signature.
 1: Verify that $r, s \in [1, n-1]$
 2: $e = H(m)$        ▷ $H$ is the same cryptographic hash function as used for signature generation.
 3: $w = s^{-1} \bmod n$
 4: $u_1 = ew \bmod n,\ \ u_2 = rw \bmod n$
 5: $X = u_1 G + u_2 Q$
 6: if $X = \mathcal{O}$ then
 7:     return (reject)
 8: end if
 9: $v = x_1 \bmod n$
10: if $v = r$ then
11:     return (accept)
12: else
13:     return (reject)
14: end if
```

Correctness of the Signature Verification

The verification of a signature (r, s) on a message m in Algorithm 6 works correctly, because, if the signature was generated by a legitimate signer, then it applies that

$$s \equiv k^{-1}(e + dr) \pmod{n}.$$

This can be rewritten as:

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}dr \pmod{n}. \tag{4.2}$$

Moreover, inserting w from Step 3 and u_1, u_2 from Step 4 in Equation (4.2), we get:

$$k \equiv we + wdr \equiv u_1 + u_2 d \pmod{n}.$$

Finally,

$$X = (x_1', y_1') = u_1 G + u_2 Q = (u_1 + u_2 d)G = kG = Q = (x_1, y_1),$$

and thus, $v = r$ since $r = x_1 \bmod n$, $v = x_1' \bmod n$.

4.3 ECDH

The *Elliptic Curve Diffie-Hellman* (ECDH) key-agreement protocol is the elliptic curve variant of the Diffie-Hellman (DH) key-agreement protocol proposed by Diffie and Hellman in [DH76]. The goal of this protocol is to establish a shared secret over an insecure channel. This is then typically used to derive a key for some symmetric cipher by using some key derivation function. The security of ECDH is based on the difficulty of the ECDHP (cf. Section 3.2).

The process of the ECDH key agreement is stated in Algorithm 7. As in the previous section, a basic prerequisite is that both participating parties, denoted by A and B, use the same domain parameters D. In the first step, A and B both compute an elliptic curve key pair, i.e., a private key d, which is a random integer in the interval $[1, n-1]$, and a corresponding public key $Q = dG$. Then, both parties exchange their public key Q_A and Q_B, respectively. Now, A computes a point $K = (x_k, y_k) = d_A Q_B$ and B computes $K' = (x_k', y_k') = d_B Q_A$. The points K and K' are equal, since

$$d_A Q_B = d_A d_B G = d_B Q_A.$$

Usually, the x-coordinate x_k is used as shared secret.

Note that the depicted version uses temporary ephemeral keys which are not necessarily authenticated. This circumstance contains the risk of a man-in-the-middle attack, since neither A nor B can verify that the received public key actually belongs to the respective counterpart. Thus, it is advisable to authenticate the public keys by other means.

In contrast, if the public key of at least one participant is static and known to the other, the risk of a man-in-the-middle attack is annihilated. However, static public keys have some other drawbacks, concerning security as well as usage. One obvious drawback is the acquisition of the counterpart's key in an authenticated way and that it has to be kept up-to-date subsequently. Another security drawback is that static keys give no key-compromise impersonation resilience. Another problem that results from the absence is the lack of forward secrecy, meaning that if a private key leaks, not only the owner can be impersonated but also all precedent sessions in which this key was involved in can be decrypted.

Algorithm 7 ECDH key agreement

```
Goal: A and B establish a session key.
1: $A$ generates an integer $d_A \in_R [1, n-1]$, computes $Q_A = d_A G$ and sends $Q_A$ to $B$.
2: $B$ does:
    2.1: Generate $d_B \in_R [1, n-1]$, compute $Q_B = d_B G$ and send $Q_B$ to $A$.
    2.2: Compute $(x_k, y_k) = d_B Q_A$
3: $A$ computes $(x_k, y_k) = d_A Q_B$
4: The shared session key is $x_k$.
```

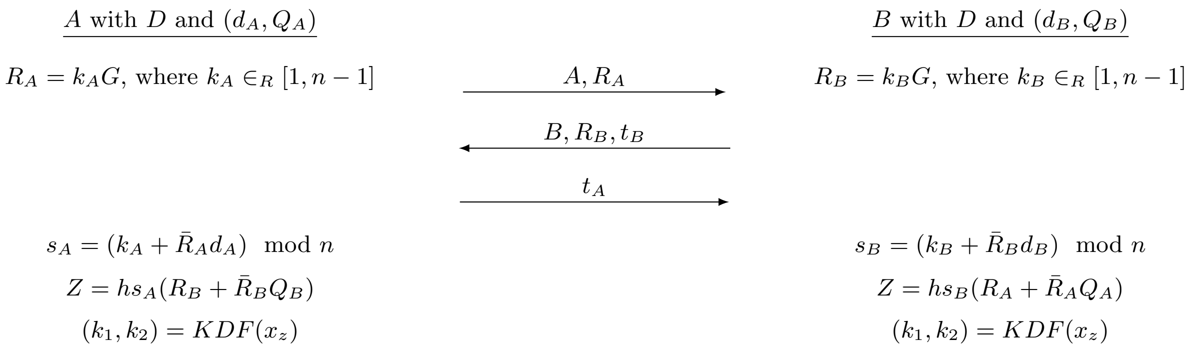

A with D and (d_A, Q_A)
B with D and (d_B, Q_B)
$R_A = k_A G$, where $k_A \in_R [1, n-1]$
A, R_A
$R_B = k_B G$, where $k_B \in_R [1, n-1]$
B, R_B, t_B
t_A
$s_A = (k_A + \bar{R}_A d_A) \mod n$
$Z = hs_A(R_B + \bar{R}_B Q_B)$
$(k_1, k_2) = KDF(x_z)$
$s_B = (k_B + \bar{R}_B d_B) \mod n$
$Z = hs_B(R_A + \bar{R}_A Q_A)$
$(k_1, k_2) = KDF(x_z)$

Figure 4.1: ECMQV key-agreement protocol

4.4 ECMQV

ECMQV is the elliptic curve variant of MQV, a three-pass protocol for key agreement based on the Diffie-Hellman scheme. The advantage of ECMQV compared to ECDH is that it is authenticated and thus, it is not susceptible to a man-in-the-middle attack. The MQV protocol was introduced in 1995 by Menezes, Qu and Vanstone in [MQV95]. Later, Law et al. [LMQ+03] improved the protocol in order to work with an arbitrary finite group, and in particular, with elliptic curve groups. ECMQV has been standardized in ISO/IEC 15946-3 [ISO02], ANSI X9.63 [ANS01], and IEEE 1363-2000 [IEE00].

The transmissions of the ECMQV protocol are depicted in Figure 4.1. The protocol in detail, based on [LMQ+03] and [HMV04], is stated in Algorithm 8. In the following, KDF denotes a key derivation function, such as `ANSI-X9.63-KDF`, and MAC denotes a secure message authentication code algorithm, such as `HMAC`. A detailed description of valid types for KDF and MAC can be found in [Cer09]. For the process of computing a shared secret Z in the ECMQV protocol, it is essential to know the following. $f = \lfloor \log_2 n \rfloor + 1$ denotes the bitlength of n, where n is the prime order of the base point G. Furthermore, if $R = (x, y)$ is a finite elliptic-curve point, then $\bar{R}$ is defined as the integer $(\bar{x} \mod 2^{\lceil f/2 \rceil}) + 2^{\lceil f/2 \rceil}$, where $\bar{x}$ is the integer representation of x.

For the stated version of the protocol it is presumed that both parties have agreed on the domain parameters D, and that A and B are in possession of a valid elliptic curve keypair (d_A, Q_A) and (d_B, Q_B), respectively. Furthermore, it is also presumed that A has an authentic copy of B's longterm public key Q_B and vice versa. For ECMQV it applies that the protocol run terminates with failure if any verification fails.

As we have mentioned in the beginning of this section, the ECMQV protocol is based on the ECDH key agreement described in Section 4.3. Thus, the authors of [HMV04] call it an "extension to the ordinary Diffie-Hellman key-agreement protocol". They also point out that the quantities s_A and s_B, computed in Step 3.2 and Step 2.3 of Algorithm 8, serve as *implicit signatures* for the ephemeral public keys R_A and R_B, respectively. s_A can be viewed as a signature, since only A is able to compute it, and also as implicit since B verifies the validity indirectly, when deriving the shared secret Z in Step 2.3, by using

$$s_A G = (k_A G + \bar{R}_A d_A G) = R_A + \bar{R}_A Q_A.$$

The same applies for s_B, the implicit signature for R_B. Unlike the ordinary Diffie-Hellman key agreement, where the shared secret would be $Z = k_A k_B G$, the shared secret in ECMQV

Algorithm 8 ECMQV key agreement

```
Goal: $A$ and $B$ establish a session key.
1: $A$ generates an integer $k_A \in_R [1, n-1]$, computes $R_A = k_A G$ and sends $A, R_A$ to $B$.
2: $B$ does:
    2.1: Perform an embedded public-key validation of $R_A$ (see Section 4.2.2).
    2.2: Generate $k_B \in_R [1, n-1]$ and compute $R_B = k_B G$.
    2.3: Compute $s_B = (k_B + \bar{R}_B d_B) \mod n$ and $Z = hs_B(R_A + \bar{R}_A Q_A)$.
         Verify that the shared secret $Z \neq \mathcal{O}$.
    2.4: Use a $KDF$ to derive two shared keys from the $x$-coordinate of $Z$:
         $(k_1, k_2) = KDF(x_z)$.
    2.5: Compute $t_B = MAC_{k_1}(2, B, A, R_B, R_A)$ and send $B, R_B, t_B$ to A.
3: $A$ does:
    3.1: Perform an embedded public-key validation of $R_B$ (see Section 4.2.2).
    3.2: Compute $s_A = (k_A + \bar{R}_A d_A) \mod n$ and $Z = hs_A(R_B + \bar{R}_B Q_B)$.
         Verify that the shared secret $Z \neq \mathcal{O}$.
    3.3: Use a $KDF$ to derive two shared keys from the $x$-coordinate of $Z$:
         $(k_1, k_2) = KDF(x_z)$.
    3.4: Compute $t = MAC_{k_1}(2, B, A, R_B, R_A)$ and verify that $t = t_B$.
    3.5: Compute $t_A = MAC_{k_1}(3, A, B, R_A, R_B)$ and send $t_A$ to B
4: $B$ computes $t = MAC_{k_1}(3, A, B, R_A, R_B)$ and verifies that $t = t_A$.
5: The shared session key is $k_2$.
```

is $Z = hs_A s_B G$. The multiplication by h and the check that $Z \neq \mathcal{O}$ afterwards, ensure that Z is of prime order n and, hence, $Z \in \langle G \rangle$.

To distinguish the authentication tags created by the initiator A and the responder B, the strings "2" and "3" are included in the MAC inputs in the Steps 2.5, 3.4, and 3.5. The verification of the authentication tags t_A and t_B has three functions. A successful verification assures that the other party has indeed computed the shared secret Z, since computing such a tag requires knowledge of the key k_1, which can only be derived from Z. It ensures that the communication has not been tampered with, as long as the MAC is secure. And in addition, the entities know the identity of the entity they are communicating with, since the identities are included in the MAC input.

Note that the ECMQV protocol has been dropped from NSA's Suite B set of cryptographic standards. Furthermore, since the ECMQV protocol has some weaknesses, Sarr et al. proposed an improved ECMQV key agreement with additional security properties in [SEB09], called Fully Hashed Menezes-Qu-Vanstone (FHMQV) protocol.

4.5 ECIES

The *Elliptic Curve Integrated Encryption Scheme* (ECIES) is a hybrid encryption scheme and a variant of the ElGamal public-key encryption scheme [ElG85]. A hybrid encryption scheme combines an asymmetric and a symmetric encryption system. The asymmetric system is used to encapsulate the key under which the message is encrypted with a symmetric encryption system. This offers good performance due to the symmetric encryption without the requirement that the sender and receiver have to share a common secret beforehand.

ECIES was originally proposed in 1997 by Bellare and Rogaway [BR97]. Since then, it has been modified several times and was finally standardized in ANSI X9.62 [ANS01] and ISO/IEC 18033-2 [ISO06].

4.5.1 Encryption

The ECIES encryption procedure, as it is stated in Algorithm 9, uses the following cryptographic primitives:

- $ENC_{k_1}(m)$ is an arbitrary symmetric encryption scheme where k_1 is the key under which the input m is encrypted. The corresponding decryption function is denoted by DEC_{k_1}.
- KDF denotes a hash-based key derivation function, e.g., `ANSI-X9.63-KDF` [ANS01]. Such a KDF can generate a key of arbitrary length ℓ. Therefore, in the $KDF(x)$ the hashes $H(x, i)$ are concatenated until a hash value of ℓ bits has been generated. The value i represents a counter that is incremented for every execution of H.
- MAC is a message authentication code, e.g., the 160 bit `HMAC-SHA-1-160`.

Besides the domain parameters D, another prerequisite for the ECIES encryption is the public key Q of the intended receiver. Q is required to be a key of a key pair suitable for elliptic curve cryptography, which consists of a randomly selected private key $d \in [1, n-1]$ and the corresponding public key $Q = dG$. The two symmetric keys k_1, k_2 are derived from the Diffie-Hellman shared secret Z. The key k_1 is then used to encrypt the plaintext via the symmetric encryption function ENC and the key k_2 is used to authenticate the output of ENC via the message authentication code MAC, which thwarts chosen-ciphertext attacks, since an adversary cannot generate valid ciphertexts on his own. Finally, the resulting ciphertext (R, C, t) on the plaintext m consists of the sender's one-time public key R, the ciphertext C and the authentication code t of C.

Note that, as stated in [HMV04], it is required to derive the symmetric keys (k_1, k_2) from x_Z as well as from the sender's one-time public key R in order to ensure that the scheme does not become malleable. If R is not included, an adversary could replace R with $-R$ in the ciphertext (R, C, t) and would obtain a different, but valid ciphertext with the same underlying plaintext as the original ciphertext.

Algorithm 9 ECIES encryption

```
Input: Domain parameters $D = (q, FR, S, a, b, G, n, h)$, public key $Q$, plaintext $m$.
Output: Ciphertext $(R, C, t)$.
1: Select $k \in_R [1, n-1]$
2: $R = kG$ and $Z = hkQ$
3: if $Z = \mathcal{O}$ then
4:     go to Step 1
5: end if
6: $(k_1, k_2) = KDF(x_Z, R)$                ▷ $x_Z$ is the $x$-coordinate of $Z$
7: $C = ENC_{k_1}(m)$ , $t = MAC_{k_2}(C)$
8: return $(R, C, t)$
```

4.5.2 Decryption

The required input for the ECIES decryption are the domain parameters D, the ciphertext (R, C, t), and the private key d that is corresponding to the public key Q used for encryption. The first step is to check whether R is a valid public key. The reason, why this is important, is described in Section 4.2. The whole decryption process of a ciphertext (R, C, t) is detailed in Algorithm 10. There, KDF denotes the same key derivation function that was used for the encryption (Algorithm 9). The same applies for the message authentication code MAC in Step 4. DEC denotes the decryption function that corresponds to the encryption function ENC, as described in Section 4.5.1.

Algorithm 10 ECIES decryption

```
Input: Domain parameters $D = (q, FR, S, a, b, G, n, h)$, private key $d$, ciphertext $(R, C, t)$.
Output: Decryption of $(R, C, t)$ (or rejection of the ciphertext).
 1: Perform an embedded public key validation of $R$ (see Section 4.2.2).
 2: $Z' = hdR$
 3: if $Z' = \mathcal{O}$ then
 4:     return (reject)
 5: end if
 6: $k_1, k_2 = KDF(x'_Z, R)$                ▷ $x'_Z$ is the $x$-coordinate of $Z'$
 7: $t' = MAC_{k_2}(C)$
 8: if $t' \neq t$ then
 9:     return (reject)
10: end if
11: $m = DEC_{k_1}(C)$
12: return $m$
```

Correctness of the Decryption

The decryption of a ciphertext (R, C, t) that was generated by a legitimate entity on a plaintext m works correctly, since

$$hdR = hd(kP) = hk(dP) = hkQ. \tag{4.3}$$

The encrypter computes the symmetric keys $(k_1, k_2) = KDF(x_Z, R)$, where $(x_Z, y_Z) = hkQ$ and the decrypter computes $(k_1, k_2) = KDF(x'_Z, R)$, where $(x'_Z, y'_Z) = hdR$. Thus, by Equation (4.3) both parties compute the same two keys (k_1, k_2).

4.6 Summary

The chapter gave insight into four cryptographic protocols on elliptic curves. In the beginning, it explained what domain parameters are and why they are necessary for the presented protocols. We illustrated the ECDSA signature generation and verification and touched on the public key validation in general. We also discussed the ECDH key agreement, a fundamental scheme when it comes to key agreements. On the basis of ECDH, the ECMQV key agreement was examined and its protocol transmissions were detailed. The last part of this chapter dealt with ECIES, a hybrid encryption scheme based on the ElGamal public-key scheme.

Chapter 5

Scalar Multiplication Algorithms

This chapter, based on [HMV04] and [CFA$^+$05], considers various scalar multiplication algorithms. On elliptic curves, a scalar multiplication, also called point multiplication, is the multiplication of a point on the curve by a (large) scalar in order to obtain another point on the same curve. Such multiplication is performed by repeated point addition operations and point doublings. Scalar multiplication is the most time-consuming operation in cryptographic schemes on elliptic curves. Hence, it is important that scalar multiplications can be efficiently computed. In this chapter, we are going to examine different methods that provide such efficient computations. The methods presented do not use any special structure of the elliptic curve.

The first two sections consider multiplication methods for computing kP, where k is a positive integer and P is a point on an elliptic curve defined over $\mathbb{F}_q$. The very basic, yet not really practical, multiplication methods are considered in Section 5.1. A more advanced method for computing kP is examined in Section 5.2. Section 5.3 covers multiplication methods where the point P is fixed, as it is the case for ECDSA signature generation. These methods make use of precomputation data that solely depend on P, in order to decrease execution time. In the last part of this chapter, Section 5.4 deals with so-called multiple point multiplication techniques, which provide an efficient computation of $kP + lQ$, which is, for example, the most costly part of the ECDSA signature verification. Finally, the chapter is summarized in Section 5.5.

5.1 Basic Point Multiplication Methods

This section gives an overview of some very basic techniques for scalar multiplication. Although they are intrinsically not really practical, these techniques serve as a basis for the more advanced multiplication methods that follow later in this chapter.

5.1.1 The Left-to-right Binary Method

The *left-to-right binary method* as well as its counterpart, the right-to-left binary method, are the most basic methods for point multiplication on elliptic curves. They are the additive versions of the basic repeated-square-and-multiply methods for exponentiation. Thus, we are not going to detail the right-to-left method due to its similarity to the left-to-right version. Because of its simplicity the left-to-right method will be described only very briefly in the following.

To obtain kP, where $k = (k_{t-1}, \dots, k_1, k_0)$ is a positive integer in unsigned binary representation with $t \approx m = \lceil \log_2 q \rceil$, and P is a point on $E(\mathbb{F}_q)$, the scalar k of bitsize t is processed from left to right. For every nonzero bit in k, the point P is added to the designated result Q. Between each of these steps, Q is doubled to ensure the correct place value of the next bit to process. This method is shown in Algorithm 11.

Algorithm 11 Left-to-right binary method

```
Input: $k = (k_{t-1}, \dots, k_1, k_0)_2 \in \mathbb{Z}_{>0}$, $P \in E(\mathbb{F}_q)$.
Output: $kP$.
1: $Q = \mathcal{O}$
2: for $i = t - 1$ downto 0 do
3:     $Q = 2Q$
4:     if $k = 1$ then
5:         $Q = Q + P$
6:     end if
7: end for
8: return $Q$
```

The binary representation of k has a density of approximately $\frac{t}{2} \approx \frac{m}{2}$. Thus, Algorithm 11 requires m point doubling operations denoted by `D` and roughly $\frac{m}{2}$ point addition operations denoted by `A` on average. As a consequence, it has an expected running time of

$$\frac{m}{2}\mathtt{A} + m\mathtt{D},$$

where additions are in general more expensive than doublings. In order to reduce the density of nonzero digits in the representation of the scalar k, and thus reduce the number of required additions, the so-called non-adjacent form can be used instead of the basic unsigned binary representation.

5.1.2 The Non-adjacent Form

The *non-adjacent form* (NAF) representation is a signed digit representation where no two consecutive digits are nonzero. Every positive integer k has a unique NAF representation denoted by NAF(k). The integer k in the non-adjacent form of length ℓ is represented as

$$k = \sum_{i=0}^{\ell-1} k_i 2^i \quad \text{where} \quad k_i \in \{0, \pm 1\}, \ k_{\ell-1} \neq 0.$$

Note that NAF(k) has the least number of nonzero digits of all signed digit representations of k. If the length of the binary representation of k is m, then the length ℓ of NAF(k) is at most $m+1$. For the length ℓ, it is true that $\frac{2^\ell}{3} < k < \frac{2^{\ell+1}}{3}$. Among all NAFs of length ℓ, the average density of nonzero digits is approximately $1/3$.

Example 5.1.1. Consider an integer $k = 7$ and let $\bar{\nu}$ denote the negative integer $-\nu$. The binary representation of k is $(0\ 1\ 1\ 1)_2$, calculated in decimal as $4 + 2 + 1 = 7$. The NAF representation of k, denoted by NAF(7), is $(1\ 0\ 0\ \bar{1})_{\mathrm{NAF}}$, calculated in decimal as $8 - 1 = 7$.

Since subtraction is as efficient as addition on elliptic curves, such a signed representation is of great interest as it allows a reduction of required additions due to the lower density of nonzero digits compared to the binary representation. The non-adjacent form of an integer k can be efficiently computed with Algorithm 12, where the NAF representation is obtained by repeatedly dividing k by 2 with remainder $u_i \in \{0, \pm 1\}$. If k is odd, the remainder u is chosen so that $(k - u_i)/2$ is divisible by 2 to ensure that the next digit in the NAF is 0.

Algorithm 12 Computing the NAF representation

```
Input: $k \in \mathbb{Z}_{>0}$.
Output: $\mathrm{NAF}(k) = (u_{t-1}, \dots, u_1, u_0)_2$ with $u_i \in \{0, \pm 1\}$ .
1: $i = 0$
2: while $k \geq 1$ do
3:     if $k \equiv 1 \mod 2$ then
4:         $u_i = 2 - k \pmod 4$
5:         $k = k - u_i$
6:     else
7:         $u_i = 0$
8:     end if
9:     $k = k/2$
10:    $i = i + 1$
11: end while
12: return $(u_{i-1}, u_{i-2}, \dots, u_1, u_0)$
```

Algorithm 13 uses NAF(k) for computing kP. It is a modification of the very basic left-to-right binary method for point multiplication (Algorithm 11). In this algorithm, the NAF(k)-representation is processed bitwise from left to right. Depending on the digit's sign, for each nonzero digit, the point P is added to or subtracted from the result stored in point Q. In every step, Q is doubled in order to update the current place value within the NAF. Thus, Algorithm 13 has an expected running time of

$$\frac{m}{3}\mathtt{A} + m\mathtt{D}.$$

Algorithm 13 NAF point multiplication

```
Input: $k \in \mathbb{Z}_{>0},\ P \in E(\mathbb{F}_q)$.
Output: $kP$.
1: $\mathrm{NAF}(k) = \sum_{i=0}^{\ell-1} u_i 2^i$.          ▷ (using Algorithm 12)
2: $Q = \mathcal{O}$
3: for $i = \ell - 1$ to 0 do
4:     $Q = 2Q$
5:     if $u_i = 1$ then
6:         $Q = Q + P$
7:     else if $u_i = -1$ then
8:         $Q = Q - P$
9:     end if
10: end for
11: return $Q$
```

To reduce the expected running time of computing kP, a window method can be used, where in each step w digits of k are processed.

5.2 The Windowed NAF Method

The number of additions for computing kP can be reduced at the costs of additional memory by using the windowed NAF method.

To do so, k has to be represented in form of a width-w NAF (also NAF_w or WNAF), where w is commonly referred to as the *window size* or *width.* WNAF(k) is a representation $k = \sum_{i=0}^{\ell-1} u_i 2^i$, where among w consecutive digits at most one coefficient is nonzero. Each of these nonzero coefficients $u_i \in [-2^{w-1}+1, 2^{w-1}-1]$ is odd and $u_{\ell-1} \neq 0$. Every positive integer k has a unique WNAF representation. The length ℓ of WNAF(k) is at most $t+1$, where t denotes the length of the binary representation of k. A WNAF(k)-representation of length ℓ has an approximate density of nonzero digits of $1/(w+1)$. Note that for $w = 2$, WNAF(k) equals NAF(k).

Example 5.2.1. Let $k = 678575$ and let $\bar{x}$ denote the negative integer $-x$. Then, the unsigned binary representation of k is

$$k = (1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1)_2,$$

and the corresponding WNAF(k)-representation for $w = 4$ is

$$k = (5\ 0\ 0\ 0\ 3\ 0\ 0\ 0\ 0\ \bar{5}\ 0\ 0\ 0\ \bar{5}\ 0\ 0\ 0\ \bar{1})_{\mathrm{NAF}_w}.$$

In [Sol00], Solinas describes the process of obtaining a WNAF-representation as a window of width w that is moved along the binary representation of the scalar k from right to left. The content of each window is used to output the next entry of WNAF(k). Algorithm 14 shows in detail how the WNAF(k)-representation can be obtained efficiently. The integer k is repeatedly divided by 2. If k is odd, the remainder $u_i = k \bmod\mathrm{s}\ 2^w$. Here, *mods* returns the smallest residue in absolute value, i.e., $u_i \equiv k \pmod{2^w}$, where $u_i \in [-2^{w-1}, 2^{w-1}-1]$. Since k is odd in this case, the remainder u_i is odd, too. This ensures that $(k - u_i)/2$ will be divisible by 2^{w-1} and, thus, the next $w-1$ digits are 0.

Algorithm 14 Computing the width-w NAF representation

```
Input: Window width $w$, $k \in \mathbb{Z}_{>0}$.
Output: $\mathrm{NAF}_w(k)$.
 1: $i = 0$
 2: while $k \geq 1$ do
 3:     if $k \equiv 1 \mod 2$ then
 4:         $u_i = k$ mods $2^w$
 5:         $k = k - u_i$
 6:     else
 7:         $u_i = 0$
 8:     end if
 9:     $k = k/2$
10:     $i = i + 1$
11: end while
12: return $(u_{i-1}, u_{i-2}, \ldots, u_1, u_0)$
```

Algorithm 15 illustrates how a WNAF(k)-representation is used for point multiplication. It is based on the NAF multiplication given in Algorithm 12. At the beginning, iP for all $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$ is precomputed and stored in a table T, where T_i denotes the i-th element of T. After WNAF(k) is obtained, it is processed from left to right. Similar to the NAF multiplication, each step requires at least one doubling operation. If $u_i \neq 0$, the precomputed value T_{u_i} is looked up and added to Q or subtracted from Q, depending on the sign of u_i. Hence, this algorithm has an expected running time of

$$\left[1\mathtt{D} + \left(2^{w-2} - 1\right)\mathtt{A}\right] + \left[\frac{m}{w+1}\mathtt{A} + m\mathtt{D}\right].$$

The first term $\left[1\mathtt{D} + \left(2^{w-2} - 1\right)\mathtt{A}\right]$ is the expected running time of the precomputation step. It is composed of one doubling operation to obtain $2P$ and $2^{w-2} - 1$ point additions with $2P$ to compute $\left[2^{w-1} - 1, \ldots, 5, 3\right] P$. It is possible to do the precomputation step beforehand for points known a priori, such as base points. This reduces the expected running time of Algorithm 15 to

$$\left[\frac{m}{w+1}\mathtt{A} + m\mathtt{D}\right].$$

Algorithm 15 Window NAF point multiplication

Input: Window width w, $k \in \mathbb{Z}_{>0}$, $P \in E(\mathbb{F}_q)$.
Output: kP.

```
 1: Precompute: $T = iP$ for $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
 2: $NAF_w(k) = \sum_{i=0}^{\ell-1} u_i 2^i$                    ▷ (using Algorithm 14)
 3: $Q = \mathcal{O}$
 4: for $i = \ell - 1$ downto $0$ do
 5:     $Q = 2Q$
 6:     if $u_i \neq 0$ then
 7:         if $u_i > 0$ then
 8:             $Q = Q + T_{u_i}$
 9:         else
10:             $Q = Q - T_{-u_i}$
11:         end if
12:     end if
13: end for
14: return $Q$
```

5.3 Fixed Point Multiplication Methods

This section describes the *fixed-base comb method* and the *fixed-base comb method with two tables*. These methods are due to Lim and Lee [LL94] and they are two of the fastest scalar multiplication methods available. As both require intense precomputations, they are commonly used for multiplications with a fixed point only, i.e., a point known a priori, such as a generator of an elliptic curve group.

5.3.1 The Fixed-base Comb Method

In the *fixed-base comb method*, the unsigned binary representation of a scalar k of a maximum bitsize t, where t is the curve's group order in bits, is considered to represent a binary matrix. This matrix consists of w rows and d columns, where w denotes the window size and $d = \lceil t/w \rceil$. Therefore, the bit representation of k is prepended with 0s, such that the length of k is a multiple of w, and then split into w blocks K_i with $0 \leq i < w$ of size d, so that

$$k = K_{w-1} \| \cdots \| K_1 \| K_0.$$

Thus, each bitstring K_i represents one row of this $w \times d$ binary matrix:

$$\begin{bmatrix} K_0 \\ \vdots \\ K_i \\ \vdots \\ K_{w-1} \end{bmatrix} = \begin{bmatrix} K_{0,d-1} & \cdots & K_{0,0} \\ \vdots & & \vdots \\ K_{i,d-1} & \cdots & K_{i,0} \\ \vdots & & \vdots \\ K_{w-1,d-1} & \cdots & K_{w-1,0} \end{bmatrix} = \begin{bmatrix} k_{d-1} & \cdots & k_0 \\ \vdots & & \vdots \\ k_{(i+1)d-1} & \cdots & k_{id} \\ \vdots & & \vdots \\ k_{wd-1} & \cdots & k_{(w-1)d} \end{bmatrix}$$

Here, $K_{j,i}$ denotes the i-th bit of the bitstring K_j and k_i denotes the i-th bit of the unsigned binary representation of the scalar k. The columns of this matrix are then processed one at a time.

By precomputing the points

$$[a_{w-1}, \ldots, a_2, a_1, a_0]P = a_{w-1}2^{(w-1)d}P + \cdots + a_2 2^{2d}P + a_1 2^d P + a_0 P,$$

for all possible bit strings $[a_{w-1}, \ldots, a_1, a_0]$, the actual computation can be accelerated. Algorithm 16 shows the precomputation in detail.

Algorithm 16 Precomputation for the fixed-base comb method

```
*Precompute* (compute $[a_{w-1}, \ldots, a_0]P$ for all bitstrings $(a_{w-1}, \ldots, a_0)$ of length $w$):
**Input:** Window width $w$, $d = \lceil t/w \rceil, P \in E(\mathbb{F}_q)$.
**Output:** Table $T$ holding the precomputed points.
 1: $Q = P$
 2: $T = (P, \mathcal{O})$
 3: for $i = 1$ to $w - 1$ do
 4:     $Q = 2^d Q$
 5:     for $j = 0$ to $2^i - 1$ do
 6:         $T_{2^i+j} = Q + T_j$
 7:     end for
 8:     $T = (T_{2^{i+1}-1}, \ldots, T_{2^i}, T)$
 9: end for
10: return $T$
```

The precomputation has expected costs of

$$(2^w - 2)\,\mathtt{A} + ((w-1)d)\,\mathtt{D}.$$

The multiplication of the fixed-base comb method, using the precomputed table T, is given in Algorithm 17.

Algorithm 17 Multiplication routine of the fixed-base comb method

```
Input: Window width $w$, $d = \lceil t/w \rceil, k = (k_{t-1}, \ldots, k_1, k_0)_2$, lookup table $T$ generated via
    Algorithm 16 with respect to $P \in E(\mathbb{F}_q)$.
Output: $kP$.
1: If necessary, pad $k$ on the left with 0s, interpret $k$ as $K_{w-1}\|\cdots\|K_1\|K_0$, where each
   $K_j$ is a length $d$ bitstring, where $K_{j,i}$ denotes the $i$-th bit of $K_j$.
2: $Q = \mathcal{O}$.
3: for $i = d-1$ downto 0 do
4:     $Q = 2Q$
5:     $Q = Q + [K_{w-1,i}, \ldots, K_{1,i}, K_{0,i}]_2 P = Q + T_{\sum_{j=0}^{w-1} K_{j,i} 2^j}$
6: end for
7: return $Q$
```

Algorithm 17 has an expected running time of

$$\left(\frac{2^w - 1}{2^w} d - 1\right) \mathtt{A} + (d-1)\mathtt{D}.$$

For the multiplication, the fixed-base comb method requires approximately as many point doublings as point additions. To reduce the number of doublings, the so-called fixed-base comb method with two tables has been introduced.

5.3.2 The Fixed-base Comb Method with Two Tables

The *fixed-base comb method with two tables* makes use of a second precomputation table in order to reduce the number of required point doublings approximately by half. This additional table has the sames size as the first table and contains the precomputations

$$2^e[a_{w-1}, \ldots, a_2, a_1, a_0]P = a_{w-1}2^{e+(w-1)d}P + \cdots + a_2 2^{e+2d}P + a_1 2^{e+d}P + a_0 P,$$

where $e = \lceil d/2 \rceil$. Algorithm 18 details the required computations to obtain the two tables.

Algorithm 18 Precomputation for the fixed-base comb method with two tables

```
Precompute (compute $[a_{w-1}, \ldots, a_0]_2 P$ and $2^e[a_{w-1}, \ldots, a_0]_2 P$ for all bitstrings
$(a_{w-1}, \ldots, a_0)$ of length $w$):
Input: Window width $w$, $d = \lceil t/w \rceil, e = \lceil d/2 \rceil, P \in E(\mathbb{F}_q)$
Output: Tables $T, T'$ holding the precomputed points.
1: $Q = P, \quad Q' = 2^e Q$
2: $T = (P, \mathcal{O}), \quad T' = (Q', \mathcal{O})$
3: for $i = 1$ to $w-1$ do
4:     $Q = 2^d Q, \quad Q' = 2^e Q$
5:     for $j = 0$ to $2^i - 1$ do
6:         $T_{2^i+j} = Q + T_j, \quad T'_{2^i+j} = Q' + T'_j$
7:     end for
8:     $T = (T_{2^{i+1}-1}, \ldots, T_{2^i}, T), \quad T' = (T'_{2^{i+1}-1}, \ldots, T'_{2^i}, T')$
9: end for
10: return $(T, T')$
```

The expected costs for computing T and T' are

$$2\,(2^w - 2)\,\mathtt{A} + ((e+d)(w-1))\,\mathtt{D}.$$

Algorithm 19 shows the multiplication using this additional table. Here, the d columns of the binary matrix are split into a lower and an upper half, each of size e. For each step in the algorithm, the point $[K_{w-1,i}, \dots, K_{1,i}, K_{0,i}]P$ is looked up in the first precomputation table and the point $[K_{w-1,i+e}, \dots, K_{1,i+e}, K_{0,i+e}]P$ is looked up in the second table. By adding both points to Q at once, the number of required doubling operations is approximately reduced by half.

Algorithm 19 Multiplication routine of the fixed-base comb method with two tables

```
Input: Window width $w$, $d = \lceil t/w \rceil, e = \lceil d/2 \rceil, k = (k_{t-1}, \dots, k_1, k_0)_2$, lookup tables $T$
    and $T'$ generated via Algorithm 18 with respect to $P \in E(\mathbb{F}_q)$.
Output: $kP$.
1: If necessary, pad $k$ on the left with 0s, interpret $k$ as $K_{w-1}\| \cdots \|K_1\|K_0$, where each
   $K_j$ is a length $d$ bitstring, where $K_{j,i}$ denotes the $i$-th bit of $K_j$.
2: $Q = \mathcal{O}$.
3: for $i = e-1$ downto $0$ do
4:     $Q = 2Q$
5:     $Q = Q + [K_{w-1,i}, \dots, K_{1,i}, K_{0,i}]P + 2^e[K_{w-1,i+e}, \dots, K_{1,i+e}, K_{0,i+e}]P =$
         $= Q + T_{\sum_{j=0}^{w-1} K_{j,i}2^j} + T'_{\sum_{j=0}^{w-1} K_{j,i+e}2^j}$
6: end for
7: return $Q$
```

A multiplication using the fixed-base comb method with two tables has an expected running time of approximately

$$\left(\frac{2^w - 1}{2^w} 2e - 1\right) \mathtt{A} + (e-1)\,\mathtt{D}.$$

The two-table method in Algorithm 19 gives a benefit over Algorithm 17 whenever

$$\frac{2^{w-1}(w-2)}{2^w - w - 1} \geq \frac{\mathtt{A}}{\mathtt{D}}.$$

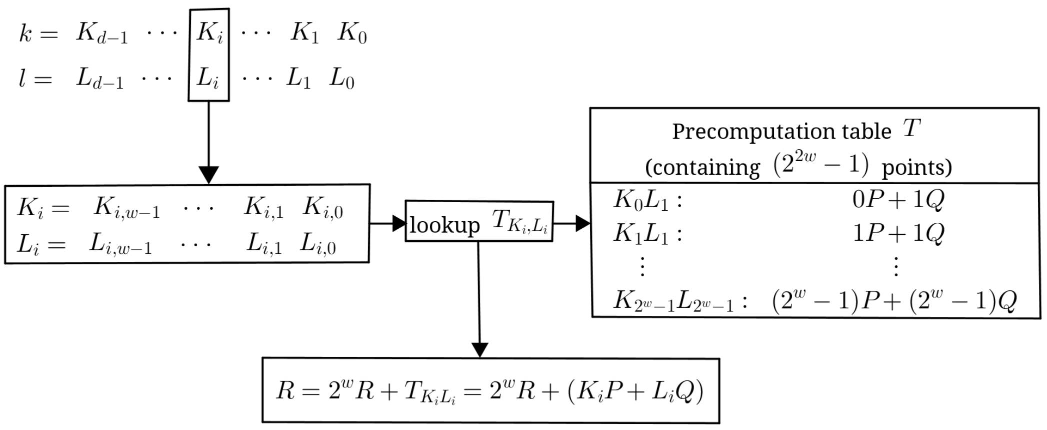
5.4 Multiple Point Multiplication Methods

In this section, we are going to discuss two different algorithms for multiple point multiplication. Firstly, we have a look at the simultaneous multiple point multiplication algorithm, which provides a speed up for calculation $kP + lQ$. Secondly, we examine the interleaving method, which provides a fast computation of the sum of v points each multiplied by some scalar k_j, such that $Q = \sum_{j=1}^{v} k_j P_j$.

5.4.1 The Simultaneous Multiple Point Multiplication Method

The *simultaneous multiple point multiplication method* offers a speedup for the computation $kP + lQ$, which is a computationally expensive step in the ECDSA signature verification (see Section 4.2). Instead of first computing kP and lQ separately and then

$kP + lQ$, the Straus-Shamir trick ([ElG85], [Str64]) is used to carry out these three steps simultaneously. Therefore, k and l are seen as two t-bit integers in unsigned binary representation. Together, they are represented as a $2 \times t$ matrix, called the *exponent array.* This matrix is processed column-wise from left to right, where for each column, both rows are processed at once using a lookup table T with precomputations. Table T contains all possible values $iP + jQ$ for $i, j \in \{0, 1\}$. Every column contains two values K_i, L_i, which are interpreted as the value of the i-th bit of k and l, respectively. Thus, in each step the value for $K_iP + L_iQ$ is looked up in the precomputation table T and added to the designated result. Between each column this result is doubled to ensure the correct place value. Hence, each step or the processing of each column, respectively, requires one addition and one doubling operation, which is not very efficient.



$k = K_{d-1} \cdots K_i \cdots K_1 \; K_0$
$l = L_{d-1} \cdots L_i \cdots L_1 \; L_0$
$K_i = K_{i,w-1} \cdots K_{i,1} \; K_{i,0}$
$L_i = L_{i,w-1} \cdots L_{i,1} \; L_{i,0}$
lookup T_{K_i,L_i}
Precomputation table T
(containing $(2^{2w}-1)$ points)
K_0L_1 : $0P+1Q$
K_1L_1 : $1P+1Q$
$K_{2^w-1}L_{2^w-1}$: $(2^w-1)P+(2^w-1)Q$
$R = 2^wR + T_{K_iL_i} = 2^wR + (K_iP + L_iQ)$

Figure 5.1: Accumulation step of the simultaneous multiple point multiplication method

To reduce the number of additions, the exponent array can be processed using a window of size w. This means, in each step w columns of the matrix are processed. Hence, the array is now processed in $d = \lceil t/w \rceil$ steps, requiring a table T with $2^{2w} - 1$ precomputed points $iP + jQ$ for all $i, j \in [0, 2^w - 1]$, as it is depicted in Figure 5.1. Thus, a reduction of additions can be achieved at the cost of increasing the effort for the precomputation, and as a consequence, an increased required storage for T. When using a window to process the exponent array, K_i and L_i are interpreted as length w bitstrings, and $K_{i,j}$ and $L_{i,j}$ denote the i-th bit of K_i and L_i, respectively. The explicit simultaneous multiple point multiplication operation using a window of size w is detailed in Algorithm 20.

Algorithm 20 Simultaneous multiple point multiplication

```
Input: Window width $w$, $d = \lceil t/w \rceil$, $k = (k_{t-1}, \dots, k_1, k_0)_2 \in \mathbb{Z}_{>0}$, $l = (l_{t-1}, \dots, l_1, l_0)_2 \in \mathbb{Z}_{>0}$, $P, Q \in E(\mathbb{F}_q)$.
Output: $kP + lQ$.
1: Precompute: $S = iP$, $S' = iQ$ for $i \in [0, 2^w - 1]$.
2: Compute $T_{i,j} = S_i + S'_j$ for all $i, j \in [0, 2^w - 1]$.
3: If necessary, pad $k$ and $l$ on the left with 0s, interpret $k$ as $K_{d-1} \| \dots \| K_1 \| K_0$ and $l$ as $L_{d-1} \| \dots \| L_1 \| L_0$, where each $K_i$, $L_i$ is a length $w$ bitstring.
4: $R = \mathcal{O}$
5: for $i = d - 1$ downto $0$ do
6:     $R = 2^w R$
7:     $R = R + (K_i P + L_i Q) = R + T_{K_i, L_i}$
8: end for
9: return $R$
```

For the precomputation, T requires a storage size of $2^{2w} - 1$ points. The expected running time of Algorithm 20, including the precomputation, is approximately

$$\left[(3 \cdot 2^{2(w-1)} - 2^{w-1} - 1)\mathtt{A} + (2^{2(w-1)} - 2^{w-1})\mathtt{D}\right] + \left[\left(\frac{2^{2w} - 1}{2^{2w}} d - 1\right) \mathtt{A} + (d - 1) w \mathtt{D}\right],$$

where the first term is the expected running time of the precomputation step and the second term gives the approximate costs of the multiply step.

5.4.2 The Interleaving Multiple Point Multiplication Method

Another technique that can be employed for computing $kP + lQ$ simultaneously is the *interleaving multiple point multiplication method.* Different to the simultaneous multiple point multiplication method, the precomputations, here, do not involve a combination of the points. Each precomputed value depends only on a single point. Moreover, this method provides a fast summation of v points, each multiplied by a different scalar k_j. In order to compute $\sum_{j=1}^{v} k_j P_j$ different multiplication methods can be used, as long as the doubling step can be done jointly. With interleaving, it is possible to combine, for example, WNAF methods (cf. Section 5.2) with different window sizes, or do some point multiplications with comb methods (cf. Section 5.3.1).

But, as mentioned in [HMV04], it has to be considered that the number of required doublings for this method is determined by the method that has to perform the most doublings among the used methods. Thus, mixing comb methods with other methods is not recommended, since the benefits of the comb methods may be lost due to the number of doublings required by the other methods. An interleaving method using the WNAF method with different window sizes is detailed in Algorithm 21.

Algorithm 21 Interleaving with NAFs

```
Input: Number of points $v$, $k_j \in \mathbb{Z}_{>0}$, widths $w_j$, and points $P_j$, where $1 \leq j \leq v$.
Output: $\sum_{j=1}^{v} k_j P_j$.
 1: Precompute: $P_{j,i} = iP_j$ for $i \in \{1, 3, 5, \ldots, 2^{w_j-1} - 1\}$
 2: $NAF_{w_j}(k_j) = \sum_{i=0}^{\ell_j-1} u_{j,i} 2^i, 1 \leq j \leq v$        ▷ (using Algorithm 14)
 3: $\eta = \max\{\ell_j : 1 \leq j \leq v\}$
 4: If necessary, pad $u_j$ in the left with 0s to ensure that $\ell_j = \eta$. Interpret $u_j$ as a length
    $\ell_j$ bitstring. where $u_{j,i}$ denotes the $i$-th bit of $u_j$ and $1 \leq j \leq v$.
 5: $Q = \mathcal{O}$
 6: for $i = \eta - 1$ downto 0 do
 7:     $Q = 2Q$
 8:     for $j = 1$ to $v$ do
 9:         if $u_{j,i} \neq 0$ then
10:             if $u_{j,i} > 0$ then
11:                 $Q = Q + P_{j,u_{j,i}}$ else $Q = Q - P_{j,-u_{j,i}}$
12:             end if
13:         end if
14:     end for
15: end for
16: return $Q$
```

The algorithm computes $\sum_{j=1}^{v} k_j P_j$ using WNAF(k_j) with window size w_j. Therefore, the values iP_j for $i \in \{1, 3, 5, \ldots, 2^{w_j-1} - 1\}$ and $1 \leq j \leq v$ are precomputed, requiring $\sum_{j=1}^{v} 2^{w_j-2}$ points of storage. The precomputation for each point P_j is the same as the precomputation of the WNAF multiplication method. After obtaining WNAF(k_j) with window size w_j, these WNAFs are processed jointly from left to right. Thus, only a single doubling of the accumulator Q is required in each step. When processing the WNAFs, the precomputation value for each $u_{j,i}$ is looked up and added to Q or subtracted from Q, respectively. Algorithm 21 has an expected running time of

$$\left[|\{j : w_j > 2\}| \, \mathtt{D} + \sum_{j=1}^{v} (2^{w_j-2} - 1)\mathtt{A} \right] + \left[\max_{\ell \leq j \leq v} \ell_j \mathtt{D} + \sum_{j=1}^{v} \frac{\ell_j}{w_j + 1} \mathtt{A} \right],$$

where l_j denotes the length of WNAF(k_j) with window size w_j.

5.5 Summary

This chapter gave an overview of different scalar multiplication methods. It detailed some basic methods in order to provide an idea of the main concept of scalar multiplication. On this basis, more advanced multiplication algorithms, designed for various purposes, were discussed. These include the WNAF method as well as fixed point multiplication methods and multiple point multiplication methods. All of the methods presented in this chapter have in common that they are generic in the sense that they do not exploit any special structure of the curve.

Chapter 6

Koblitz Curves

This chapter provides a close look at the so-called binary Koblitz curves. We discuss the important properties of these curves in general and how these curves can be used to increase the speed of scalar point multiplications. Moreover, we show how certain common scalar multiplication methods can be modified to benefit from the provided speedup.

This chapter comprises four sections. In Section 6.1 the properties of Koblitz curves are detailed along with all curve specific prerequisites for the subsequent sections. In addition, this section gives a basic idea of how these properties affect the curves' security levels. Section 6.2 illustrates different ways of representing scalars in order to make use of these speedups and how they are efficiently obtained. Section 6.3 deals with those scalar multiplication methods introduced in Chapter 5 for which adaptions to Koblitz curves are commonly known. The adaptions of the methods from Chapter 5 which are not covered here, require an alternative form of scalar representation and are described in detail in Chapter 7. Finally, in Section 6.4, the chapter is briefly summarized. This chapter is mainly based on [Sol00], [HMV04], and [CFA$^+$05]. Especially the techniques concerning solely Koblitz curves are almost all due to Solinas [Sol00].

6.1 Properties

In 1991, Neal Koblitz proposed a family of elliptic curves with arithmetic properties which offer a speedup for scalar multiplication [Kob91]. These *anomalous binary curves* are defined by Equation (2.10) stated in Section 2.3.3. The curve equation is defined over $\mathbb{F}_2$ and thus, $a_2 \in \{0, 1\}$ and $a_6 = 1$, leading to the equation

$$E_a : y^2 + xy = x^3 + a_2x^2 + 1. \tag{6.1}$$

Note that due to the fact that a_6 is always 1 for binary Koblitz curves, it is common practice to denote a_2 just as a when used in context of Koblitz curves.

6.1.1 Group Order

The group of these curves E_a is defined over $\mathbb{F}_{2^m}$ with m prime, and is denoted as $E_a(\mathbb{F}_{2^m})$. As stated in [MVO96], for cryptographic reasons, the group order should be a large prime or *almost prime*. Almost prime group order means that it is the product of a large prime and a small integer. The only Koblitz curve groups over $\mathbb{F}_2$ are

$$\begin{aligned} E_0(\mathbb{F}_2) &= \{\mathcal{O}, (0,1), (1,0), (1,1)\} \\ E_1(\mathbb{F}_2) &= \{\mathcal{O}, (0,1)\}. \end{aligned}$$

As $\mathbb{F}_2$ is a subfield of $\mathbb{F}_{2^m}$, the group $E(\mathbb{F}_2)$ is a subgroup of $E(\mathbb{F}_{2^m})$. Hence, the order $\#E_a(\mathbb{F}_{2^m})$ is always divisible by the order of $E_a(\mathbb{F}_2)$. $\#E_a(\mathbb{F}_2)$ is called the *cofactor* of the group order, commonly denoted as h or f, which is

$$h = \#E_a(\mathbb{F}_2) = \begin{cases} 4 & \text{if } \ a = 0 \\ 2 & \text{if } \ a = 1. \end{cases} \tag{6.2}$$

Due to Equation (6.2), it is obvious that Koblitz curves have an almost prime group order.

6.1.2 Frobenius Endomorphism

As we have already seen, the curve equations of Koblitz curves are defined over $\mathbb{F}_2$. Thus, if a point $P = (x, y)$ is on the curve E_a, then Equation (6.1) also holds for the point $P' = (x^2, y^2)$, and, hence, P' is on E_a too. Due to the Freshman's dream (see Section 2.1.4), the mapping

$$\tau(x, y) = (x^2, y^2), \ \ \tau(\mathcal{O}) = \mathcal{O} \tag{6.3}$$

is a homomorphism, i.e., it applies that $\tau(P + Q) = \tau(P) + \tau(Q)$ for all $P, Q \in E_a(\mathbb{F}_{2^m})$. Moreover, as $\tau(P) \in E_a(F_{2^m})$, $\tau : E_a(\mathbb{F}_{2^m}) \to E_a(\mathbb{F}_{2^m})$ is an endomorphism on this particular curve type and is called *Frobenius endomorphism.*

We know that

$$(x^4, y^4) + 2(x, y) = \mu \cdot (x^2, y^2) \ \text{ for all } \ (x, y) \in E_a, \ \text{where} \tag{6.4}$$

$$\mu = (-1)^{1-a}. \tag{6.5}$$

Writing Equation (6.4) in terms of the Frobenius map (6.3), we get

$$(\tau^2 + 2)P = \mu\tau(P) \ \text{ for all } \ P \in E_a(\mathbb{F}_{2^m}), \ \text{where}$$

$\tau^l(P)$ denotes the l-fold application of τ to P. Hence, the Frobenius map can be regarded as a complex number τ satisfying the characteristic polynomial

$$\tau^2 + 2 = \mu\tau. \tag{6.6}$$

Due to the fact that squaring is quite inexpensive on binary curves, the costs for applying τ are insignificant compared to other operations. Through Equation (6.6), the relation $2P = \mu\tau(P) - \tau^2(P)$ holds for all $P \in E(\mathbb{F}_{2^m})$ which suggests that one can exchange doublings for applications of τ. Yet, so far this relation does not give us an advantage, as doubling is exchanged for two applications of τ and an expensive addition. The solution to this problem is to recode the scalar from base 2 to base τ and thereby trade each of the expensive point doubling operations for an application of τ, as we are going to see later.

6.1.3 The ECDLP on Koblitz Curves

Wiener and Zuccherato [WZ98] discovered that curves equipped with a Frobenius endomorphism allow a speedup for solving the elliptic curve discrete logarithm problem (ECDLP). Similar work has been done independently by Gallant, Lambert, and Vanstone [GLV00].

On Koblitz curves the parallelized Pollard's rho collision search algorithm (cf. Section 3.3.1) can be sped up by a factor of $\sqrt{2m}$. This is achieved by taking equivalence classes into account, which are due to the Frobenius endomorphism, and the negation map.

Consider $\langle P \rangle$ as the subgroup of prime order n of $E_a(\mathbb{F}_{2^m})$ generated by point P. The Frobenius map ϕ, where ϕ^m is the identity map and thus, $\tau^m \equiv 1 \pmod{n}$, induces the partitioning of this subgroup into equivalence classes using an equivalence relation on $\langle P \rangle$. The equivalence relation on $\langle P \rangle$ is defined as $S \sim T$ if $S = \pm\tau^i T$ for $i \in \{0, 1, \dots, m-1\}$. Due to this relation, on Koblitz curves the prime order subgroup of the curve can be partitioned into $(n-1)/2m$ equivalence classes of size $2m$ and one additional class containing $\mathcal{O}$. This reduces the search space of the parallelized Pollard's rho algorithm by a factor of $2m$, which in the following leads to a reduction of the expected running time by a factor of $\sqrt{2m}$. This speedup for the parallelized Pollard's rho algorithm to solve the ECDLP on Koblitz curves is acceptable regarding the curve's security level, as it lowers it only by a few bits.

For example, consider the NIST-recommended elliptic curve `K-233`. On this Koblitz curve over the binary field $\mathbb{F}_{2^{233}}$ the conventional parallelized Pollard's rho method requires approximately 2^{116} operations and therefore it meets the security level of 112 bits. When exploiting the Frobenius map together with the negation map on this curve, the expected running time is reduced by a factor of $\sqrt{2 \cdot 233}$. Thus, the parallelized Pollard's rho algorithm then requires approximately 2^{112} operations. Hence, despite this reduction the curve still retains the security level of 112 bits. For more detailed information on this topic, we additionally refer the reader to [HMV04].

6.1.4 Lucas Sequences for τ

Since we will need Lucas sequences for the complex number τ numerous times in the subsequent part of this chapter, we are going to show here how these are defined.

We have already described the Lucas sequences in general in Section 2.1.6. We know that they can be generated over quadratic equations and that the parameters p and q for Lucas sequences are derived from the root of the characteristic polynomial.

The complex number τ satisfies Equation (6.6) and, thus, it follows that the parameter $p = \mu$, where μ is given in Equation (6.5), and $q = 2$. Hence, the first Lucas sequence U_k for τ is defined as

$$U_0 = 0,\ U_1 = 1 \ \text{ and } \ U_k = \mu U_{k-1} - 2U_{k-2} \ \text{ for } k \geq 2. \tag{6.7}$$

Accordingly, the second Lucas sequence V_k for τ is

$$V_0 = 0,\ V_1 = \mu \ \text{ and } \ V_k = \mu V_{k-1} - 2V_{k-2} \ \text{ for } k \geq 2. \tag{6.8}$$

6.2 τ-adic Representations

In order to take advantage of the Frobenius endomorphism for scalar multiplication, we have to convert scalars from base 2 to base τ, which results in a so-called τ-adic representation. In this section, we are going to take a closer look at the *τ-adic NAF* representation and its enhanced version, the *reduced τ-adic NAF*.

Before we have a look at the conversion of scalars, we have to introduce the norm function in $\mathbb{Z}[\tau]$. The ring $\mathbb{Z}[\tau]$ is Euclidean with respect to $N(\cdot)$, i.e., it is possible to perform a division with remainder in $\mathbb{Z}[\tau]$ so that the norm of the remainder is smaller than the norm of the divisor. More precisely, the norm $N(\alpha) \in \mathbb{Z}$ of an element $\alpha \in \mathbb{Z}[\tau]$ is the integer product of α and its complex conjugate $\bar{\alpha}$.

The norm function is multiplicative, i.e., $N(\alpha\beta) = N(\alpha)N(\beta)$ for all $\alpha, \beta \in \mathbb{Z}[\tau]$. Furthermore, $N(\pm 1) = 1$ and $N(\alpha) \geq 0$, where $N(\alpha)$ equals 0 if and only if $\alpha = 0$. The norm $N(\alpha)$ of an element $\alpha = a_0 + a_1\tau \in \mathbb{Z}[\tau]$ is given by

$$N(\alpha) = (a_0 + a_1\tau)(a_0 + a_1\bar{\tau}) = a_0^2 + \mu a_0 a_1 + 2a_1^2 \in \mathbb{Z},$$

where the complex number $\tau = \frac{\mu+\sqrt{-7}}{2}$ is the first root of Equation (6.6) and its conjugate $\bar{\tau}$ is the second root, namely $\bar{\tau} = \frac{\mu-\sqrt{-7}}{2}$. Hence, it is easy to verify that

$$N(\tau) = \tau\bar{\tau} = 2 \quad \text{and} \quad N(\tau - 1) = h, \tag{6.9}$$

where h is the cofactor given in Equation (6.2), and also that

$$\tau + \bar{\tau} = \mu, \tag{6.10}$$

where μ is given in Equation (6.5). In addition, $N(\tau^m - 1) = \#E_a(\mathbb{F}_{2^m})$. Due to the fact that the norm is multiplicative, it follows that $N((\tau^m - 1)/(\tau - 1)) = \#E_a(\mathbb{F}_{2^m})/h = n$.

In the following, divisibility by τ is important for deriving a τ-adic representation of a scalar. An element $\alpha = a_0 + a_1\tau \in \mathbb{Z}[\tau]$ is divisible by τ if and only if a_0 is even. If that is the case, then $\alpha/\tau = (a_1 + \mu a_0/2) - (a_0/2)\tau$.

6.2.1 The τ-adic Non-adjacent Form (TNAF)

Similar to the NAF-form of positive integers (cf. Section 5.1.2), there is a unique τ-adic NAF for every element of $\mathbb{Z}[\tau]$. Due to the property of the norm function that $N(\tau) = 2$, any positive integer k can be represented in the form $k' = \sum_{i=0}^{l-1} u_i\tau^i$ with $u_i \in \{0, \pm 1\}$.

This so-called TNAF representation is obtained by repeatedly dividing a scalar k, represented as $\kappa \in \mathbb{Z}[\tau]$, by τ with remainder $u_i \in \{0, \pm 1\}$, for which $N(u_i) < N(\tau)$ with $N(u_i) \in \{0, 1\}$ holds. To ensure that no two consecutive digits are nonzero, Solinas showed that an element $\kappa = k_0 + k_1\tau \in \mathbb{Z}[\tau]$ is not only divisible by τ if and only if k_0 is even, but also that κ is divisible by τ^2 if and only if $k_0 \equiv 2k_1 (\mathrm{mod}\ 4)$. Thus, if $\kappa \in \mathbb{Z}[\tau]$ is not divisible by τ then $u_i \in \{0, \pm 1\}$ is chosen so that $(\kappa - u_i)/\tau$ is divisible by τ to ensure that the next digit in the TNAF is a 0. The TNAF of an integer can be efficiently obtained via Algorithm 22.

Solinas showed in [Sol00] that the length $l(k)$ of a TNAF(k) is bounded by

$$\log_2(N(k)) - 0.54627 < l(k) < \log_2(N(k)) + 3.5156 \tag{6.11}$$

when $l(k) > 30$, and that among all TNAFs of length l the average density of nonzero digits is $\approx 1/3$.

Algorithm 22 Computing the τ-adic NAF [HMV04, Algorithm 3.61].

```
Input: $\kappa = k_0 + k_1\tau \in \mathbb{Z}[\tau]$
Output: TNAF($\kappa$) = $(u_{t-1}, \ldots, u_1, u_0)_\tau$ with $u_i \in \{0, \pm 1\}$ .
1: $i = 0$
2: while $(r_0 \neq 0 \parallel r_1 \neq 0)$ do
3:     if $r_0 \equiv 1 \mod 2$ then
4:         $u_i = 2 - (k_0 - 2k_1 \mod 4)$
5:         $k_0 = k_0 - u_i$
6:     else
7:         $u_i = 0$
8:     end if
9:     $tmp = k_0,\ k_0 = \mu k_0$
10:    $k_0 = k_1 + k_0/2,\ k_1 = -tmp/2$
11:    $i = i + 1$
12: end while
13: return $(u_{i-1}, u_{i-2}, \ldots, u_1, u_0)$
```

According to Equation (6.11), the length of a TNAF(k) is approximately $log_2(N(k))$ which is equal to $2\log_2(k)$, and is twice the length of NAF(k). The Hamming weight H of a length-l NAF(k) is approximately $\frac{l}{3}\log_2 k$, which applies also to the τ-adic NAF. Consequently, the Hamming weight of TNAF(k) is twice the Hamming weight of NAF(k). Hence, replacing the NAF(k) by the TNAF(k) will eliminate doubling operations in the scalar multiplication methods, but will double the number of additions. To overcome this problem, Solinas introduced the *reduced τ-adic NAF* in [Sol00].

6.2.2 The Reduced τ-adic Non-adjacent Form (RTNAF)

In order to obtain a reduced τ-adic NAF, the goal is to find an element $\rho \in \mathbb{Z}[\tau]$ with $\rho \equiv k \mod ((\tau^m - 1)/(\tau - 1))$ and smallest norm possible, in order to compute TNAF(ρ) which has approximately the length of the binary NAF(k).

Solinas showed that if γ is an element of $\mathbb{Z}[\tau]$ with $\gamma \equiv k \mod (\tau^m - 1)$ then $\gamma P = kP$ for all $P \in E_a(\mathbb{F}_{2^m})$ because $(\tau^m - 1)(P) = \tau^m(P) - P = P - P = \mathcal{O}$. He also showed that if an element $\rho \equiv k \pmod{\delta}$, where $\delta = \frac{\tau^m - 1}{\tau - 1}$, then $\rho P = kP$ with respect to the main subgroup of order n. In other words, $\rho P = kP$ for all points P of order n in $E_a(\mathbb{F}_{2^m})$ and, thus, TNAF(ρ) is equivalent to TNAF(k) for all points of order n, where TNAF(ρ) has approximately half the length of TNAF(k) and, therefore, has approximately the length of the binary NAF(k). For the required modulo operation in $\mathbb{Z}[\tau]$, it is necessary to perform a division of two elements in $\mathbb{Z}[\tau]$.

Division in $\mathbb{Z}[\tau]$

So, for the reduced τ-adic NAF it is essential to find an element $\rho \in \mathbb{Z}[\tau]$ with $\rho \equiv k \pmod{\delta}$ and minimal norm. Therefore, we are going to briefly explain how to obtain a quotient $\kappa \in \mathbb{Z}[\tau]$ and a remainder $\rho \in \mathbb{Z}[\tau]$ with $\alpha = \kappa\beta + \rho$ and $N(\rho)$ as small as possible for any $\alpha, \beta \in \mathbb{Z}[\tau]$ with $\beta \neq 0$, where it is assured that $N(\rho) < N(\beta)$.

$$\lambda = \frac{\alpha}{\beta} = \frac{\alpha\bar{\beta}}{N(\beta)} = \frac{(a_0 + a_1\tau)(b_0 + b_1\bar{\tau})}{N(\beta)}.$$

We know that $\tau\bar{\tau} = N(\tau) = 2$. So, we obtain

$$\lambda = \frac{a_0b_0 + a_0b_1\bar{\tau} + 2a_1b_1 + a_1b_0\tau}{b_0^2 + \mu b_0b_1 + 2b_1^2}.$$

Now, to get rid of $\bar{\tau}$ in the numerator, we can replace $a_0b_1\bar{\tau}$ with $\mu a_0b_1 - a_0b_1\tau$ because, as we know from Equation 6.10, $\tau + \bar{\tau} = \mu$ and, thus, $a_0b_1\tau + a_0b_1\bar{\tau} = \mu a_0b_1$. We obtain

$$\lambda = \frac{(b_0 + \mu b_1)a_0 + 2a_1b_1 + (a_1b_0 - a_0b_1)\tau}{b_0^2 + \mu b_0b_1 + 2b_1^2}$$

and this can be written as

$$\lambda = \frac{g_0 + g_1\tau}{N} = \frac{g_0}{N} + \frac{g_1}{N}\tau \;=\; \lambda_0 + \lambda_1\tau,$$

where $g_0 = a_0b_0 + \mu a_0b_1 + 2a_1b_1$ and $g_1 = a_1b_0 - a_0b_1$.

The quotient κ is then the element in $\mathbb{Z}[\tau]$ close to the complex number λ. To find such an element κ that is close to the complex number $\lambda = \lambda_0 + \lambda_1\tau$ with $\lambda_0, \lambda_1 \in \mathbb{Q}$, we do a "rounding-off" via the algorithm Solinas stated in [Sol00]. In the following, we use *Round(λ_0, λ_1)* to denote this rounding algorithm. The resulting remainder ρ of the division satisfies $\rho \equiv \alpha \pmod{\beta}$. For the detailed division algorithm that performs these steps, we refer the reader to [Sol00].

Reduction of an Integer

For the RTNAF, we have to reduce an integer k modulo $(\tau^m - 1)/(\tau - 1)$. This can be achieved via a modification of the previously mentioned division in $\mathbb{Z}[\tau]$ algorithm. Recall that $\delta = (\tau^m - 1)/(\tau - 1)$ has norm n. On the basis of the division in $\mathbb{Z}[\tau]$, we set the dividend α to k, i.e., $\alpha = k + 0\tau$, and the divisor δ is given in canonical form $d_0 + d_1\tau$. This leads to $\lambda = \frac{g_0}{N} + \frac{g_1}{N}\tau = \frac{a_0d_0+\mu a_0d_1}{N(\delta)} + \frac{-a_0d_1}{N(\delta)}\tau = \frac{k(d_0+\mu d_1)}{N(\delta)} + \frac{k(-d_1)}{N(\delta)}\tau$. We introduce two auxiliary values s_0 and s_1, where $s_0 = d_0 + \mu d_1$ and $s_1 = -d_1$. When applying these values to the previous equation, we get

$$\lambda = \frac{g_0}{N(\delta)} + \frac{g_1}{N(\delta)}\tau = \frac{s_0k}{n} + \frac{s_1k}{n}\tau.$$

Note that $s_0,\ s_1$ can be computed efficiently via the Lucas sequence U_k (see Section 6.1.4) involving μ and the cofactor h:

$$s_i = \frac{(-1)^i}{h}(1 - \mu U_{m+3-a-i}).$$

Using these modifications, we obtain Algorithm 23 that can be used to reduce an integer modulo $(\tau^m - 1)/(\tau - 1)$.

Algorithm 23 Reduction modulo $\delta = (\tau^m - 1)/(\tau - 1)$

```
Input: Integer $k \in [1, n-1]$, $s_0 = d_0 + \mu d_1$, $s_1 = -d_1$, where $\delta = d_0 + d_1\tau$, $N = N(\delta)$.
Output: The integers $r_0$, $r_1$ specifying $r_0 + r_1\tau = k \pmod{(\tau^m - 1)/(\tau - 1)}$
1: $d_0 = s_0 + \mu s_1$
2: $\lambda_0 = s_0 k/N$
3: $\lambda_1 = s_1 k/N$
4: $(q_0, q_1) = Round(\lambda_0, \lambda_1)$   ▷ (using the "rounding-off"-Algorithm mentioned before)
5: $r_0 = k - d_0 q_0 + 2 s_1 q_1$
6: $r_1 = s_1 q_0 - s_0 q_1$
7: return $(r_0, r_1)$
```

A disadvantage of this reduction method is the need for two multiprecision divisions by $N(\delta)$. To circumvent this problem, Solinas proposed a *partial reduction* algorithm in [Sol00], which is discussed subsequently.

Partial Reduction

The partial reduction algorithm yields an element $\rho' \in \mathbb{Z}[\tau]$ that is congruent to k modulo δ, but unlike the result ρ of Algorithm 23, ρ' does not necessarily have the smallest possible norm. The two rational numbers λ_0, λ_1 in the complex number λ, which require the multiprecision divisions, are replaced by *approximations* λ'_0, λ'_1. More precisely, Solinas replaced the multiprecision divisions in Step 2 and Step 3 of Algorithm 23 with two multiplications by values of bitsize $\frac{m+5}{2} + C$. The constant $C \in \mathbb{N}$, denotes the number of bits of accuracy of the approximations. The choice of C gives a trade-off between the costs of computing λ'_i and the probability that λ'_i equals λ_i. More precisely, the bigger C is, the more expensive is the computation of λ_i, but also the higher are the chances that the result of the partial reduction equals the result of the full reduction.

Algorithm 24 shows this partial reduction. It is based on the algorithm given in [HMV04] which is slightly more efficient than the one Solinas published in [Sol00], although the versions are very similar.

Algorithm 24 Partial reduction modulo $\delta = (\tau^m - 1)/(\tau - 1)$

```
Input: Integer $k \in [1, n-1]$, $C \geq 2$, $s_0 = d_0 + \mu d_1$, $s_1 = -d_1$, where $\delta = d_0 + d_1\tau$.
Output: Integers $r_0, r_1$ satisfying $r_0 + r_1\tau = k$ partmod $\delta$.
1: $k' = \lfloor k/2^{a-C+(m-9)/2} \rfloor$
2: $V_m = 2^m + 1 - \#E_a(\mathbb{F}_{2^m})$
3: $\alpha = 2^{(m+5)/2}$, $\beta = 2^C$
4: for $i = 0$ to $1$ do
5:     $g' = s_i k'$
6:     $j' = V_m \lfloor g'/2^m \rfloor$
7:     $\lambda_i = \lfloor (g' + j')/\alpha + \frac{1}{2} \rfloor / \beta$
8: end for
9: $(q_0, q_1) = \mathrm{Round}(\lambda_0, \lambda_1)$   ▷ (using the "rounding-off"-Algorithm mentioned before)
10: $r_0 = k - (s_0 + \mu s_1) q_0 - 2 s_1 q_1$
11: $r_1 = s_1 q_0 - s_0 q_1$
12: return $(r_0, r_1)$
```

Solinas proved that the $TNAF(\rho)$ has a maximum length of $m+a$ and $TNAF(\rho')$ has a length of at most $m+a+3$ if $C \geq 2$. He also stated that there is the possibility that $\rho' \neq \rho$, but with C large enough it is insignificantly small. More precisely, the probability that $\rho' \neq \rho$ is bounded by $2^{-(C-5)}$. So, by reducing the scalar k using Algorithm 24 before transforming it into a TNAF via Algorithm 22, we get a reduced τ-adic NAF.

6.3 τ-adic Scalar Multiplication Methods

In this section, we are going to show how the scalar multiplication methods explained in Chapter 5 have to be modified in order to replace the doubling operations by more efficient applications of the Frobenius endomorphism. The section is mainly based on [Sol00], [CFA+05], and [HMV04].

6.3.1 τ-adic NAF Point Multiplication Methods

Subsequently, we deal with the τ-adic versions of the two common scalar multiplication methods: *NAF* and *windowed NAF*.

The τ-adic NAF Method

The non-adjacent form multiplication method is, as already mentioned in Section 5.1, a basic multiplication method. Although this method is not often used in practical applications, it gives a good picture of how the modification of the scalar multiplication methods towards τ-adic versions works in general.

At first, the representation of the scalar k has to be changed from base 2 to base τ. In this case here, we need k to be represented as τ-adic NAF. The conversion is achieved via Algorithm 22 in combination with Algorithm 24, both explained in the previous section. Secondly, having such a representation enables us to replace the doubling operation in Algorithm 13 with applications of τ. Algorithm 25 shows this τ-adic version of Algorithm 13, where in Step 1 and Step 2 the TNAF representation of the scalar k is derived, and in Step 5 the doubling operation is replaced by an application of τ.

Algorithm 25 TNAF point multiplication

```
Input: Integer $k \in [1, n-1]$, $P \in E(\mathbb{F}_q)$.
Output: $kP$.
 1: $\rho' = k$ partmod $\delta$                    ▷ (using Algorithm 24)
 2: $TNAF(\rho') = \sum_{i=0}^{l-1} u_i \tau^i$      ▷ (using Algorithm 22)
 3: $Q = \mathcal{O}$.
 4: for $i = l-1$ to $0$ do
 5:     $Q = \tau Q$
 6:     if $u_i = 1$ then
 7:         $Q = Q + P$
 8:     else if $u_i = -1$ then
 9:         $Q = Q - P$
10:     end if
11: end for
12: return $Q$
```

The Windowed τ-adic NAF Method

The width-w TNAF representation, often also called WTNAF or TNAF$_w$, is the τ-adic analogue of the ordinary width-w NAF representation used in Section 5.2 and likewise offers a speedup for the TNAF point multiplication method (Algorithm 25) in exchange for additional memory. As with the ordinary WNAF, the scalar is processed in windows of size w. For $w > 1$, every element $\kappa \in \mathbb{Z}[\tau]$ can be expressed in the form $\kappa = \sum_{i=0}^{l-1} u_i\tau^i$ where $u_{l-1} \neq 0$ and $u_i \in \{0\} \cup \{\pm\alpha_u : \alpha_u \in \pi\}$ with $\pi = \{\alpha_u \equiv u \mod \tau^w : u = 1, 3, \ldots, 2^{w-1} - 1\}$. Analogously to the ordinary WNAF, there is at most one nonzero coefficient allowed among w consecutive coefficients.

Simultaneously to the TNAF, we reduce the scalar k via Algorithm 24 before we compute WTNAF(k) to ensure that the length of the WTNAF is approximately the same as the one of WNAF(k).

After performing a partial reduction of k, the WTNAF of the resulting element $\rho \in \mathbb{Z}[\tau]$ is then obtained via Algorithm 26. In this algorithm, ρ is repeatedly divided by τ allowing a remainder $\gamma \in \{0, \pm\alpha_1, \pm\alpha_2, \ldots, \pm\alpha_{2^{w-1}-1}\}$. More precisely, if ρ is not divisible by τ, then the remainder $\gamma = \alpha_u$ with $u = \rho$ mods τ^w is chosen in such a way that $(\rho - \alpha_u)/\tau$ is then divisible by τ^{w-1}, ensuring that there is no nonzero value among the next $w - 1$ coefficients.

In order to be able to compute u, Solinas showed in [Sol00] that an integer value t_k exists that satisfies the same polynomial equation over $\mathbb{Z}/2^k\mathbb{Z}$ as τ over the complex numbers, i.e., if $r_0 + r_1\tau \equiv 0 \pmod{\tau^k}$ then $r_0 + r_1t_k \equiv 0 \pmod{2^k}$. This value t_k is defined as

$$t_k = 2U_{k-1}U_k^{-1} \pmod{2^k} \text{ for } k \geq 1,$$

where $(U_k)_{k\geq 0}$ is the first Lucas sequence for τ, stated in Equation (6.7).

Consequently, $\tau \mapsto t_w$ induces a map $\phi_w : \mathbb{Z}[\tau] \to \mathbb{Z}/2^w\mathbb{Z}$. Solinas proved that $\phi_w(\alpha) = 0$ if and only if α is divisible by τ^w, with $\alpha \in \mathbb{Z}[\tau]$. It follows that ϕ_w, which is a surjective ring homomorphism, has the kernel $\{\alpha \in \mathbb{Z}[\tau] : \tau^w \text{ divides } \alpha\}$. Thus, each congruence class of $\mathbb{Z}[\tau]/\tau^w\mathbb{Z}[\tau]$ corresponds under ϕ_k to an element of $\mathbb{Z}/2^k\mathbb{Z}$, where the odd congruence classes of $\mathbb{Z}[\tau]/\tau^w\mathbb{Z}[\tau]$ correspond to the odd elements of $\mathbb{Z}/2^k\mathbb{Z}$. Hence, the odd numbers $\{\pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)\}$ are incongruent modulo τ^w and also not divisible by τ. Accordingly, the numbers $\{\pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{(2^{w-1}-1)}\}$ where $\alpha_u = u \bmod \tau^w$ are incongruent modulo τ^w.

When the numbers α_u are known, one can obtain WTNAF(ρ), where $\rho = k$ partmod $(\tau^m - 1)/(\tau - 1)$, via Algorithm 26 as mentioned before. Note that an integer $u = r_0 + r_1t_w$ *mods* 2^w is a unique integer in $[-2^{w-1}, 2^{w-1}]$ satisfying $u \equiv r_0 + r_1t_w \pmod{2^w}$. In Table 6.1 and Table 6.2, respectively, the values of α_u and t_w are listed for window sizes $w = 3, 4, 5$.

w	t_w	u	u mod τ^w	TNAF(u mod τ^w)	α_u
3	2	1	1	(1)	1
		3	$\tau+1$	$(-1,0,-1)$	$\tau+1$
4	10	1	1	(1)	1
		3	$-\tau-3$	$(1,0,-1)$	τ^2-1
		5	$-\tau-1$	$(1,0,1)$	τ^2+1
		7	$-\tau+1$	$(1,0,0,-1)$	τ^3-1
5	26	1	1	(1)	1
		3	$-\tau-3$	$(1,0,-1)$	τ^2-1
		5	$-\tau-1$	$(1,0,1)$	τ^2+1
		7	$-\tau+1$	$(1,0,0,-1)$	τ^3-1
		9	$-2\tau-3$	$(1,0,1,0,0,1)$	$\tau^3\alpha_5+1$
		11	$-2\tau-1$	$(-1,0,-1,0,-1)$	$-\tau^2\alpha_5-1$
		13	$-2\tau+1$	$(-1,0,-1,0,1)$	$-\tau^2\alpha_5+1$
		15	$3\tau+1$	$(1,0,0,0,-1)$	$\tau^2\alpha_5-\alpha_5$

Table 6.1: Expression for α_u for $a=0$ and $3 \leq w \leq 5$

w	t_w	u	u mod τ^w	TNAF(u mod τ^w)	α_u
3	6	1	1	(1)	1
		3	$-\tau+1$	$(-1,0,-1)$	$-\tau+1$
4	6	1	1	(1)	1
		3	$\tau-3$	$(1,0,-1)$	τ^2-1
		5	$\tau-1$	$(1,0,1)$	τ^2+1
		7	$\tau+1$	$(-1,0,0,-1)$	$-\tau^3-1$
5	6	1	1	(1)	1
		3	$\tau-3$	$(1,0,-1)$	τ^2-1
		5	$\tau-1$	$(1,0,1)$	τ^2+1
		7	$\tau+1$	$(-1,0,0,-1)$	$-\tau^3-1$
		9	$2\tau-3$	$(-1,0,-1,0,0,1)$	$-\tau^3\alpha_5+1$
		11	$2\tau-1$	$(-1,0,-1,0,-1)$	$-\tau^2\alpha_5-1$
		13	$2\tau+1$	$(-1,0,-1,0,1)$	$-\tau^2\alpha_5+1$
		15	$-3\tau+1$	$(1,0,0,0,-1)$	$\tau^2\alpha_5-\alpha_5$

Table 6.2: Expression for α_u for $a=1$ and $3 \leq w \leq 5$

The values α_u in the tables above are optimized for performance. A detailed explanation of derivation of these values and their computations can be found in [Sol00].

Algorithm 26 Computing the width-w TNAF representation

```
Input: Window width $w$, $t_w$, $\alpha_u = \beta_u + \gamma_u\tau$ for $u \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$, $\rho = r_0 + r_1\tau \in \mathbb{Z}[\tau]$.
Output: TNAF$_w(\rho)$.
1: $i = 0$
2: while $(r_0 \neq 0 \parallel r_1 \neq 0)$ do
3:     if $r_0 \equiv 1 \mod 2$ then
4:         $u = r_0 + r_1 t_w$ mods $2^w$
5:         if $u > 0$ then
6:             $s = 1$
7:         else
8:             $s = -1,\ u = -u$
9:         end if
10:        $r_0 = r_1 - s\beta_u,\ r_1 = r_1 - s\gamma_u,\ u_i = s\alpha_u$
11:    else
12:        $u_i = 0$
13:    end if
14:    $tmp = r_0,\ r_0 = \mu r_0$
15:    $r_0 = r_1 + r_0/2,\ r_1 = -tmp/2$
16:    $i = i + 1$
17: end while
18: return $(u_{i-1}, u_{i-2}, \ldots, u_1, u_0)$
```

Algorithm 27 shows how scalar multiplication on Koblitz curves, using the WTNAF representation, is performed efficiently. Due to the approximate length of the WTNAF(ρ) being m and its average density of $1/(w+1)$, a scalar multiplication performed with this algorithm requires on average $2^{w-2} - 1 + \frac{m}{w+1}$ additions.

Algorithm 27 Window TNAF point multiplication

```
Input: Window width $w$, integer $k \in [1, n-1]$, $P \in E(\mathbb{F}_q)$.
Output: $kP$.
1: Precompute: $P_u = \alpha_u P$, for $u \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$.
2: $TNAF_w(k$ partmod $\delta) = \sum_{i=0}^{l-1} u_i\tau^i$.        ▷ (using Algorithms 24, 26)
3: $Q = \mathcal{O}$
4: for $i = l - 1$ downto 0 do
5:     $Q = \tau Q$
6:     if $u_i \neq 0$ then
7:         if $u > 0$ then
8:             $Q = Q + P_u$
9:         else
10:            $Q = Q - P_{-u}$
11:        end if
12:    end if
13: end for
14: return $(Q)$
```

As with WNAF, it is possible to do the precomputation step for points known a priori, such as base points, beforehand reducing the number of additions to $\frac{m}{w+1}$.

6.3.2 τ-adic Multiple Point Multiplication Methods

As we have seen in Section 5.4, multiple point multiplications methods are used to speed up the computation of terms of the form $kP + lQ$. In the ECDSA signature verification (see Section 4.2) this computation is a computationally expensive step. These methods can be adapted for the applications of the Frobenius endomorphism on Koblitz curves too. Therefore, as with the previous shown methods, some modifications are necessary, mainly concerning the representations of the precomputations.

The τ-adic Interleaved Multiple Point Multiplication Method

In the following, the interleaved multiple point multiplication method, which we have explained in Section 5.4.2, is modified in such a way that it can operate with a τ-adic representation. As we have already seen before, the interleaved point multiplication method makes use WNAF representations. Thus, in order to exchange the doubling operation with an application of τ within this multiplication method, we require the scalars to be in windowed τ-adic NAF. As a consequence, the precomputation for each point P_j is the same as for the WTNAF multiplication method.

Algorithm 28 τ-adic interleaving with NAFs

```
Input: Number of points $v$, integers $k_j$, widths $w_j$, and points $P_j$, where $1 \leq j \leq v$.
Output: $\sum_{j=1}^{v} k_j P_j$.
1: Precompute: $P_{j,i} = \alpha_i P_j$ for $i \in \{1, 3, 5, \ldots, 2^{w_j - 1} - 1\}$   ▷ ($\alpha$ from Tables 6.1,6.2, according to $w_j$)
2: $TNAF_{w_j}(k_j \text{ partmod } \delta) = \sum_{i=0}^{l_j - 1} u_{j,i} \tau^i, 1 \leq j \leq v$   ▷ (using Algorithms 24, 26)
3: $\eta = \max\{l_j : 1 \leq j \leq v\}$
4: If necessary, pad $u_j$ on the left with 0s to ensure that $l_j = \eta$. Interpret $u_j$ as a length $l_j$ bitstring, where $u_{j,i}$ denotes the $i$-th bit of $u_j$ and $1 \leq j \leq v$.
5: $Q = \mathcal{O}$
6: for $i = \eta - 1$ downto $0$ do
7:     $Q = \tau Q$
8:     for $j = 1$ to $v$ do
9:         if $u_{j,i} \neq 0$ then
10:            if $u_{j,i} > 0$ then
11:                $Q = Q + P_{j,u_{j,i}}$ else $Q = Q - P_{j,-u_{j,i}}$
12:            end if
13:        end if
14:    end for
15: end for
16: return $Q$
```

6.4 Summary

This chapter introduced the Koblitz curves and their performance advantages when it comes to scalar multiplication. It contained an explanation of the Frobenius endomorphism and why it applies to this type of elliptic curves. Within this scope, we pointed out how this endomorphism affects the security levels of Koblitz curves. Furthermore, we detailed the τ-adic representation of a scalar and its width-w adaption. In this context we discussed the importance of encoding a scalar to a τ-adic representation and gave algorithms to efficiently obtain such representations. The last part of this chapter illustrated the τ-adic versions of the NAF and WNAF multiplication method as well as the τ-adic interleaved point multiplication method.

Chapter 7

Results

This chapter introduces an improvement of the fixed-base comb methods on binary Koblitz curves which we proposed in [HW13]. It shows how in this context point doublings can be traded for applications of the Frobenius endomorphism. Therefore, we are going to introduce a scalar recoding algorithm to obtain an unsigned τ-adic representation of scalars, which allows us to exchange the doubling steps within the multiplication methods for the application of the Frobenius endomorphism. In addition, this chapter details a modification of the simultaneous multiple point multiplication method as an example of how other conventional multiplication methods can be improved on Koblitz curves.

The chapter consists of four sections. At the beginning, Section 7.1 discusses the unsigned τ-adic representation of scalars as well as the modifications to the multiplication methods that are necessary to take advantage of this scalar recoding. Section 7.2 gives some details about the software implementations of our findings. In Section 7.3 we are going to detail the estimated scalar multiplication costs of the discussed scalar multiplication algorithms and compare the corresponding timings of our software implementations on various Koblitz curves. The chapter is concluded by a short summary in Section 7.4.

7.1 Speeding Up more Scalar Multiplication Methods on Koblitz Curves

This section details how the unsigned τ-adic representation can be obtained from a scalar. Furthermore, it discusses the modifications to the fixed-base comb multiplication methods as well as the simultaneous multiple point multiplication method that are necessary to take advantage of this scalar recoding. These multiplication methods are all explained in detail in Section 5.3 and in Section 5.4.1, respectively.

7.1.1 The Unsigned τ-adic Representation

In order to make use of the advantages of the Koblitz curves (cf. Chapter 6) with the fixed-based comb methods (cf. Section 5.3.1 and Section 5.3.2), the binary scalar representation has to be modified. It is necessary to obtain an unsigned τ-adic representation of a scalar k, containing only $0s$ and $1s$, i.e., $k = \sum_{i=0}^{l-1} u_i \tau^i$, with $u_i \in \{0, 1\}$. We introduced this technique in [HW13]. Such a representation can be obtained via Algorithm 29. This algorithm is a modification of the TNAF algorithm shown in Section 6.2.1.

Algorithm 29 Computing the unsigned τ-adic representation

```
Input: $\kappa = r_0 + r_1\tau \in \mathbb{Z}[\tau]$
Output: $\kappa$ represented as bitstring $u = (u_{t-1}, \dots, u_1, u_0)_\tau$.
 1: $i = 0$
 2: while $(r_0 \neq 0 \parallel r_1 \neq 0)$ do
 3:     if $r_0 \equiv 1 \mod 2$ then
 4:         $u_i = 1,\ r_0 = r_0 - 1$
 5:     else
 6:         $u_i = 0$
 7:     end if
 8:     $tmp = r_0,\ r_0 = \mu r_0$
 9:     $r_0 = r_1 + r_0/2,\ r_1 = -tmp/2$
10:     $i = i + 1$
11: end while
12: return $(u_{i-1}, u_{i-2}, \dots, u_1, u_0)$
```

In contrast to the TNAF algorithm, the remainder u_i in Step 4 of Algorithm 29 is only allowed to be 1. As with the TNAF-representation the size of this unsigned representation is approximately twice the size of the ordinary binary representation. Hence, in order to reduce the size of the resulting representation, it is necessary to perform a reduction by $\frac{\tau^m-1}{\tau-1}$ beforehand, as explained in Section 6.2.2.

7.1.2 τ-adic Fixed-base Comb Multiplication Methods

The fixed-base comb multiplication methods are two of the fastest scalar multiplication methods available, when the precomputations can be done beforehand. Hence, it is of interest to improve them so that they can use the unsigned τ-adic representation to gain a performance benefit on Koblitz curves.

The τ-adic Fixed-base Comb Method

To exploit the advantages of the unsigned τ-adic representation on Koblitz curves, the precomputation as well as the multiplication routine of the common fixed-base comb method have to be modified. Within the precomputation, it is essential to replace the doublings by evaluations of the map τ, i.e.,

$$[a_{w-1}, \dots, a_2, a_1, a_0]_\tau P = a_{w-1}\tau^{(w-1)d}(P) + \dots + a_2\tau^{2d}(P) + a_1\tau^d(P) + a_0P. \tag{7.1}$$

As the fixed-base comb multiplication method requires intense precomputations, this measure signifies a great performance benefit. The modified precomputation is detailed in Algorithm 30. The resulting table T contains all $2^w - 1$ points corresponding to Equation (7.1).

Algorithm 30 Precomputation for the fixed-base comb method on Koblitz curves

```
Precompute (compute $[a_{w-1},\dots,a_0]_\tau P$ for all bitstrings $(a_{w-1},\dots,a_0)$ of length $w$):
Input: Window width $w$, $d = \lceil t/w \rceil, P \in E(\mathbb{F}_q)$.
Output: Table $T$ holding the precomputed points.
 1: $Q = P$
 2: $T = (P, \mathcal{O})$
 3: for $i = 1$ to $w - 1$ do
 4:     $Q = \tau^d Q$
 5:     for $j = 0$ to $2^i - 1$ do
 6:         $T_{2^i+j} = Q + T_j$
 7:     end for
 8:     $T = (T_{2^{i+1}-1}, \dots, T_{2^i}, T)$
 9: end for
10: return $T$
```

The unsigned τ-adic scalar representation of k, obtained via Algorithm 29, enables us to replace the doubling operation with an evaluation of the map τ in the multiplication routine of the comb method. This modified routine, performing an evaluation of τ in Step 4 instead of a doubling operation, is stated in Algorithm 31. Although not much effort is required to modify the multiplication method, the resulting performance gain is tremendous as we are going to see in Section 7.3.

Algorithm 31 Multiplication routine of the fixed-base comb method on Koblitz curves

```
Input: Window width $w$, $d = \lceil t/w \rceil, k = (k_{t-1},\dots,k_1,k_0)_\tau$, lookup table $T$ generated via
    Algorithm 30 with respect to $P \in E(\mathbb{F}_q)$.
Output: $kP$.
 1: If necessary, pad $k$ on the left with 0s, interpret $k$ as $K_{w-1}\|\cdots\|K_1\|K_0$, where each
    $K_j$ is a length $d$ bitstring, where $K_{j,i}$ denotes the $i$-th bit of $K_j$.
 2: $Q = \mathcal{O}$.
 3: for $i = d - 1$ downto 0 do
 4:     $Q = \tau Q$
 5:     $Q = Q + [K_{w-1,i},\dots,K_{1,i},K_{0,i}]_\tau P = Q + T_{\sum_{j=0}^{w-1} K_{j,i}2^j}$
 6: end for
 7: return $Q$
```

The τ-adic Fixed-base Comb Method with Two Tables

As we showed in Section 5.3.2, the *fixed-base comb method with two tables* uses a second table with $2^w - 1$ precomputed points. Analogous to the τ-adic fixed-base comb method, the precomputation has to be modified to enable applications of the Frobenius map later in the multiplication part. The first precomputation table is equal to the one of the τ-adic fixed-base comb method, stated in Equation (7.1). For the additional table, the doublings must again be replaced by evaluations of the map τ, which leads to the following precomputation for the second table:

$$\tau^e[a_{w-1},\dots,a_2,a_1,a_0]_\tau P = a_{w-1}\tau^{e+(w-1)d}(P) + \dots + a_2\tau^{e+2d}(P) + a_1\tau^{e+d}(P) + a_0P.$$

After building the first table, we apply τ to each element of this table $e = \lceil d/2 \rceil$ times to build the second lookup table, as is detailed in Algorithm 32. This measure provides an additional speedup as we trade one addition per element of the second table and additional $w \cdot e$ doublings for e applications of τ per element of the second table. For reasonable values of e, the latter is by far cheaper than a single point addition.

Algorithm 32 Precomputation for the fixed-base comb method with two tables on Koblitz curves

Precompute (compute $[a_{w-1}, \ldots, a_0]_\tau P$ and $\tau^e[a_{w-1}, \ldots, a_0]_\tau P$ for all bitstrings $(a_{w-1}, \ldots, a_0)$ of length w):

Input: Window width w, $d = \lceil t/w \rceil, e = \lceil d/2 \rceil, P \in E(\mathbb{F}_q)$.

Output: Tables T, T' holding the precomputed points.

```
 1: $Q = P$
 2: $T = (P, \mathcal{O}), T' = (\mathcal{O})$
 3: for $i = 1$ to $w - 1$ do
 4:     $Q = \tau^d Q$
 5:     for $j = 0$ to $2^i - 1$ do
 6:         $T_{2^i+j} = Q + T_j$
 7:     end for
 8:     $T = (T_{2^{i+1}-1}, \ldots, T_{2^i}, T)$
 9: end for
10: for $i = 1$ to $2^w - 1$ do
11:     $T' = (\tau^e T_i, T')$
12: end for
13: return $(T, T')$
```

The modified τ-adic multiplication routine using the two precomputation tables which are obtained via Algorithm 32 is shown in Algorithm 33. Similar to the comb method employing a single table, the doubling operation in Step 4 is replaced with a much cheaper evaluation of τ.

Algorithm 33 Multiplication routine of the fixed-base comb method with two tables on Koblitz curves

Input: Window width w, $d = \lceil t/w \rceil, e = \lceil d/2 \rceil, k = (k_{t-1}, \ldots, k_1, k_0)_\tau$, lookup tables T and T' generated via Algorithm 32 with respect to $P \in E(\mathbb{F}_q)$.

Output: kP.

```
1: If necessary, pad $k$ on the left with 0s, interpret $k$ as $K_{w-1} \| \cdots \| K_1 \| K_0$, where each
   $K_j$ is a length $d$ bitstring, where $K_{j,i}$ denotes the $i$-th bit of $K_j$.
2: $Q = \mathcal{O}$.
3: for $i = e - 1$ downto $0$ do
4:     $Q = \tau Q$
5:     $Q = Q + [K_{w-1,i}, \ldots, K_{1,i}, K_{0,i}]_\tau P + \tau^e[K_{w-1,i+e}, \ldots, K_{1,i+e}, K_{0,i+e}]_\tau P =$
           $= Q + T_{\sum_{j=0}^{w-1} K_{j,i} 2^j} + T'_{\sum_{j=0}^{w-1} K_{j,i+e} 2^j}$
6: end for
7: return $Q$
```

In general, the fixed-base comb method with two tables gives a benefit over the conventional comb method whenever

$$\frac{2^{w-1}(w-2)}{2^w - w - 1} \geq \frac{\mathtt{A}}{\mathtt{D}}. \tag{7.2}$$

In the τ-adic versions, the doublings costs $\mathtt{D}$ are replaced by applications of τ. Since the costs for τ are small, the right-hand side of Equation 7.2 becomes quite large. Thus, Algorithm 33 does not give an advantage over Algorithm 31 for window sizes used in practice.

7.1.3 The τ-adic Simultaneous Multiple Point Multiplication Method

In addition to the fixed-base comb methods, where the main focus of this issue lies on, we show now an example of how other conventional scalar multiplication methods can be modified so that they benefit from the unsigned τ-adic representation on Koblitz curves. For the explanation of the approach we use the example of the conventional simultaneous multiple point multiplication method described in general in Section 5.4.1.

As we have already seen, it is necessary to apply some changes in order to obtain a τ-adic version of a multiplication method. At first, the scalars k, l have to be transformed to base τ. This is achieved via Algorithm 29 in Section 7.1.1. Then, instead of precomputing $iP + jQ$ for all $i, j \in [0, 2^w - 1]$, we now compute

$$\alpha P + \beta Q \quad \text{for all } \alpha, \beta \in \{\mathcal{O}, 1, \tau, \tau+1, \tau^2, \tau^2+1, \dots, \tau^{w-1}+1\}$$

in Step 1. Furthermore, we replace the multiplication $2^w R$ in Step 6 by w applications of τ. Algorithm 34 shows this τ-adic simultaneous multiple point multiplication method where the sets $S = \alpha P$ and $S' = \beta Q$ are precomputed at the beginning. As for all of the scalar multiplication methods in this section, if P or Q is a recurrent point, e.g., a generator point, the (pre)computation of the corresponding set S or S', respectively, might be done independently of the actual multiplication.

Algorithm 34 τ-adic simultaneous multiple point multiplication method

```
Input: Window width $w$, $d = \lceil t/w \rceil$, $k = (k_{t-1}, \dots, k_1, k_0)_\tau$, $l = (l_{t-1}, \dots, l_1, l_0)_\tau$, $P, Q \in E(\mathbb{F}_q)$.
Output: $kP + lQ$.
1: Precompute: $S = \alpha P,\ S' = \beta Q$ for all $\alpha,\ \beta \in \{\mathcal{O}, 1, \tau, \tau+1, \tau^2, \tau^2+1, ..., \tau^{w-1}+1\}$
2: Compute $T_{i,j} = S_i + S'_j$ for all $i, j \in [0, 2^w - 1]$
3: If necessary, pad $k$ and $l$ on the left with 0s, interpret $k$ as $K_{d-1}\| \cdots \|K_1\|K_0$ and $l$ as
   $L_{d-1}\| \cdots \|L_1\|L_0$, where each $K_i$,$L_i$ is a length $w$ bitstring.
4: $R = \mathcal{O}$
5: for $i = d-1$ downto $0$ do
6:     $R = \tau^w R$
7:     $R = R + (K_i P + L_i Q) = R + T_{K_i, L_i}$
8: end for
9: return $R$
```

7.2 Implementation Details

In this section, we are going to detail the important implementation properties of the τ-adic comb algorithm and the reference implementations of the WTNAF method and the conventional comb algorithm. We have implemented the NIST-recommended elliptic curves `K-163`, `K-233`, `K-283`,`K-408` and `K-571`, defined in [Nat09], in Java™. For the underlying binary fields of the curves, we are using fast reductions. The values of these binary fields are represented as polynomials (cf. Section 2.2) and stored in `long`-arrays. In order to perform squarings in linear time complexity, we are using table lookups [SOOS95] and multiplications that use the windowed left-to-right comb multiplication method. This method, which is due to Lim and Lee (cf. [HMV04, LD00]), works with precomputed multiplication lookup tables. For binary field multiplications, we use windows of size $w = 4$ and for multiplications with curve parameters, we use windows of size $w = 8$. These lookup tables are cached for curve parameters and recurring intermediate values in the addition/doubling formulas. Furthermore, partial reductions are used to derive the unsigned τ-adic representation.

We use López-Dahab coordinates [LD98] with the fast formulas given by Lange and Doche in [Lan04, CFA+05] and by Higuchi and Takagi in [HT00]. Furthermore, we perform mixed additions in all implementations. Using the López-Dahab coordinate type, mixed additions take $\mathtt{8M} + \mathtt{5S}$. However, the multiplication lookup tables for two intermediate values can be used twice. That lowers the costs to $\mathtt{6M} + \mathtt{2m} + \mathtt{5S}$. The evaluation of τ requires $\mathtt{3S}$ and a point doubling, necessary for conventional comb methods, takes $\mathtt{3M}+\mathtt{5S}$.

7.3 Estimated Costs and Timings

This section provides an overview of the estimated scalar multiplication costs of the WTNAF and the τ-adic comb multiplication method which are discussed in Section 6.3.1 and Section 7.1.2. We are also going to present the corresponding timings of our software implementations, tested on an Intel Core i5-2540M platform running Ubuntu Linux 12.10/amd64 and OpenJDK 7u15/amd64 in server mode.

Table 7.1 lists the costs of squarings and multiplications with precomputed lookup tables expressed in terms of multiplications on the test platform. On this platform, we get at most $\mathtt{A} = \mathtt{8.2M}$ per mixed addition, $\mathtt{D} = \mathtt{3.5M}$ per doubling and $\mathtt{T} = \mathtt{0.3M}$ per evaluation of τ.

	$\mathbb{F}_{2^{163}}$	$\mathbb{F}_{2^{233}}$	$\mathbb{F}_{2^{283}}$	$\mathbb{F}_{2^{409}}$	$\mathbb{F}_{2^{571}}$
$\mathtt{1S} =$	0.094M	0.080M	0.077M	0.061M	0.055M
$\mathtt{1m} =$	0.643M	0.717M	0.761M	0.829M	0.939M

Table 7.1: Costs of squarings in relation to multiplications, where `S` denotes the cost of one squaring, `M` the cost of a multiplication and `m` the cost of a multiplication using cached precomputation tables of window size $w = 4$.

In the following, we compare the costs of the WTNAF and the τ-comb method based on the relative costs given in Table 7.1. The comparison with the WTNAF method is of great interest since this method exploits the Frobenius endomorphism on Koblitz curves too. So, for example, on curve `K-233` using a window size $w = 7$ the WTNAF method has an expected running time of `284.09M`, whereas the expected running time of the τ-comb method is `264.36M`. This gives an advantage of 7.46% over the WTNAF method. On curve `K-283` the WTNAF method with $w = 8$ has expected costs of `314.00M`, whereas the τ-comb method is expected to require only `283.72M`, thus, resulting in a speedup of 10.67%. For $w = 9$ on curve `K-233`, the expected running times are `238.45M` and `201.45M`, respectively.

That means a performance gain of 18.37%. Using the same window size on curve `K-409`, we get `300.53M` compared to `365.85M` resulting in a speedup of 9.49%. In Table 7.2, we compare the timings of one fixed-base scalar multiplication of the τ-comb method with the WTNAF method and the conventional comb method, discussed in 5.3.1, on various Koblitz curves using different window sizes. The table shows that the performance gain of the τ-comb method is up to 25% compared to the WTNAF method and up to 45% compared to the conventional comb method.

Curve	**Window size**	**Comb** [μs]	**τ-Comb** [μs]	**WTNAF** [μs]	**Speedup w.r.t. Comb**	**Speedup w.r.t. WTNAF**
`K-163`	w=6	276.50	209.06	241.77	**1.32x**	**1.16x**
	w=7	237.03	196.51	208.73	**1.21x**	**1.06x**
	w=8	211.65	168.13	194.55	**1.26x**	**1.16x**
	w=9	192.83	152.97	187.38	**1.26x**	**1.22x**
`K-233`	w=6	511.96	398.87	405.49	**1.28x**	**1.02x**
	w=7	455.76	341.16	386.22	**1.34x**	**1.13x**
	w=8	383.59	299.29	356.87	**1.28x**	**1.19x**
	w=9	343.87	266.21	332.61	**1.29x**	**1.25x**
`K-283`	w=6	738.62	544.43	583.54	**1.36x**	**1.07x**
	w=7	653.78	489.27	527.23	**1.34x**	**1.08x**
	w=8	582.79	430.37	509.92	**1.35x**	**1.18x**
	w=9	508.49	393.47	448.91	**1.29x**	**1.14x**
`K-409`	w=6	1581.53	1113.97	1284.27	**1.42x**	**1.15x**
	w=7	1383.75	1024.39	1110.81	**1.35x**	**1.08x**
	w=8	1203.60	889.96	1077.14	**1.35x**	**1.21x**
	w=9	1060.99	807.13	938.07	**1.31x**	**1.16x**
`K-571`	w=6	3026.80	2135.81	2280.14	**1.42x**	**1.07x**
	w=7	2627.00	1903.24	2061.80	**1.38x**	**1.08x**
	w=8	2336.93	1675.88	1865.80	**1.39x**	**1.11x**
	w=9	2048.58	1581.32	1816.99	**1.30x**	**1.15x**

Table 7.2: Comparison of the multiplication timings of the comb, the τ-comb and the WTNAF methods

7.4 Summary

This chapter introduced modifications to the fixed-base comb methods as well as the simultaneous multiple point multiplication method, which allow us to benefit from speedups available on Koblitz curves, i.e., the possibility of replacing point doublings with applications of the far more efficient Frobenius endomorphism τ. It explained how to perform scalar recoding from base 2 to an unsigned base-τ representation and provided the respective algorithm. Furthermore, the algorithms of the modified multiplication methods, which make use of the base τ representation, were detailed in this chapter. The last part of this chapter dealt with the implementations of the modified fixed-base comb methods and gave a detailed performance comparison.

Chapter 8

Conclusions

In this thesis we focused on elliptic curve cryptography in general and scalar multiplication on elliptic curves in particular. At the beginning, the preliminaries provided an introduction to the arithmetical background of elliptic curves. Based on this, we discussed the security properties of elliptic curves. Additionally, generic attacks as well as specific attacks against the ECDLP were explained in order to give an idea of how important the choice of the curve's parameters is with respect to security. Then, we detailed several cryptographic protocols on elliptic curves, more precisely, a signature scheme, two key-agreement protocols, and a hybrid encryption scheme. These protocols, like almost all other elliptic curve schemes, involve scalar multiplication operations.

The subsequent part of the thesis was devoted to scalar multiplication methods on elliptic curves. Scalar multiplication is the most expensive operation in elliptic curve schemes. We showed various multiplication methods with different scopes. Besides the rather impractical basic point multiplication methods, the respective chapter covered multiple point multiplication methods which tend to efficiently compute an expensive step in the ECDSA signature verification. We also detailed the fixed-base comb multiplication methods which achieve very high speed at the cost of precomputations and increased memory usage due to lookup tables.

With the focus on speeding up these fixed-base comb methods, we gave a comprehensive introduction to binary Koblitz curves. These curves are interesting with regard to scalar multiplication since Koblitz curves allow trading the doubling operation for an application of the so-called Frobenius endomorphism which is inexpensive compared to point doubling. We discussed the WTNAF multiplication method which exploits this endomorphism on Koblitz curves to gain a performance benefit.

In this context, we introduced an improvement to the fixed-base comb method on Koblitz curves and showed how doubling operations can be exchanged for far more efficient applications of the Frobenius endomorphism. To do so, we presented an unsigned τ-adic representation of scalars. This scalar recoding allows us to transform scalar multiplication methods, which generally use scalars in binary representation, to τ-adic versions. We detailed the scalar recoding algorithm and introduced a modified version of the common fixed-base comb multiplication method that can exploit this unsigned τ-adic representation and gains a tremendous speedup on Koblitz curves. Software implementations of our findings as well as the WTNAF method and the conventional comb method in Java™ were used for comprehensive comparisons. For one fixed-base scalar multiplication, the τ-adic comb method showed a performance improvement of up to 25% compared with the WTNAF method and an improvement of up to 42% compared with the conventional comb method. We pointed out that the implementation of the τ-comb is straightforward and requires only little effort. Furthermore, using the example of the simultaneous multiple point multiplication method, we showed how other scalar multiplication methods that employ a scalar in binary representation, can be modified to τ-adic versions. To sum up, we showed that the τ-adic fixed-base comb method we proposed is a good alternative to other fixed-base multiplication methods.

8.1 Related and Future Work

Several approaches have been proposed with focus on scalar multiplication on Koblitz curves as well as on improving the fixed-base comb multiplication method. In [ACS04] the authors proposed a fast scalar multiplication method on Koblitz curves that uses an optimized τ-adic scalar representation. This representation is obtained as a combination of telescopic sums and a single point halving operation. With this form of scalar recoding, the number of group operations required for a scalar multiplication is reduced by roughly 14% without increasing memory usage, resulting in a speedup of at least 12% over the conventional TNAF method. The representation obtained in this approach was refined in [AHP05], mainly by optimizing its Hamming weight, resulting in about 25% fewer group operations.

The authors of [AFLR12] showed how the Frobenius endomorphism τ can be efficiently combined with the GLV-method by using powers of τ. They are able to decompose scalars into smaller scalars and then apply interleaved point multiplication to them, setting new speed records with the help of a low-level implementation. The implementations of these approaches and a comparison with our proposed method are issues for future work.

The authors of [MHH12] proposed an improved fixed-base comb method for scalar multiplication. This method combines the Tsaur-Chou method [TC05] with the Lim-Lee method in order to reduce the number of required elliptic-curve point additions, resulting in a speedup of 38% compared to the Tsaur-Chou method. Adapting this new fixed-base comb multiplication method to work with the Frobenius endomorphism, as well as implementing the approach in order to compare its performance with the τ-adic comb method, is another issue for future work.

Appendix A

Definitions

A.1 Abbreviations

ANSI	American National Standards Institute
DH	Diffie-Hellman
DL	Discrete Logarithm
DLP	Discrete Logarithm Problem
DHP	Diffie-Hellman Problem
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDHP	Elliptic Curve Diffie-Hellman Problem
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECIES	Elliptic Curve Integrated Encryption Scheme
ECMQV	Elliptic Curve Menezes-Qu-Vanstone algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standards
GCD	Greatest Common Divisor
KDF	Key Derivation Function
MAC	Message Authentication Code
MOV	Menezes-Okamoto-Vanstone algorithm
MQV	Menezes-Qu-Vanstone algorithm
NAF	Non-Adjacent Form
NIST	National Institute of Standards and Technology
RSA	Rivest-Shamir-Adleman cryptosystem
RTNAF	Reduced τ-adic Non-adjacent Form
SHA	Secure Hashing Algorithm
SSSA	Semaev-Smart-Satoh-Araki attack
TNAF	τ-adic Non-Adjacent Form
WNAF	Windowed Non-Adjacent Form
WTNAF	Windowed τ-adic Non-Adjacent Form

A.2 Used Symbols

$\forall a$	forall quantifier
$\exists a$	existential quantifier
$a \in_R [1, n]$	integer a chosen uniformly at random from the set $[1, n]$
$\lceil a \rceil$	smallest integer not less than a
$\lfloor a \rfloor$	largest integer not greater than a
$a!$	product of all positive integers less or equal than a
$\sqrt{a}$	square root of a real value or a field element
$a \cdot b$	multiplication of integers, finite fields or groups
a^b	exponentiation of an integer, finite field or group
$a \mid b$	integer a divides b without remainder
$a + b$	addition of integers, finite fields or groups
$a \mid\mid b$	concatenation of the (bit) strings a and b
$a \bmod m$	remainder of a division from a by m
$a \text{ mods } m$	smallest remainder in absolute value of a division from a by m
$a \equiv b \bmod n$	a and b congruent modulo n
$\binom{a}{b}$	binomial coefficient, i.e., $\binom{a}{b} = \frac{a!}{b!(a-b)!}$
$a_1, \ldots, a_6$	Weierstrass coefficients
$\mathtt{A}$	costs of an addition of two elliptic curve points
$\mathbb{A}^n$	affine n space
$char(K)$	characteristic of field K
$deg(f)$	degree of a polynomial f
E	elliptic curve in affine or projective Weierstrass form
$E(K)$	group of K-rational points on an elliptic curve
$\#E(K)$	number of points in $E(K)$
ΔE	discriminant of elliptic curve E
$e(P, Q)$	pairing of P and Q
$\mathbb{F}$	finite field
$\mathbb{F}_q$	finite field with q elements
$\mathcal{G}$	a group
$ord(\mathcal{G})$	order of group $\mathcal{G}$
$ord_{\mathcal{G}}(x)$	order of group element x
$\langle g \rangle$	subgroup generated by g
$\gcd(a, b)$	greatest common divisor of the integers a and b
K	a field
K^*	field K without 0, i.e., $K^* = K \setminus \{0\}$
$(k_{t-1}, \ldots, k_0)_2$	binary expansion of $k \in \mathbb{N}$
L/K	field extension of field K
$\log_b(a)$	logarithm of a to base b
$\log_G(P)$	discrete logarithm of point P with respect to generator G
$\mathtt{M}$	costs of a field multiplication
$\mathbb{N}$	set of natural numbers
$\mathcal{O}$	point at infinity on an elliptic curve
$\mathbb{P}^n$	projective n space
$\mathbb{R}$	set of real numbers
$\mathtt{S}$	costs of a field squaring
$\lvert S \rvert$	cardinality of set S

τ	Frobenius endomorphism
$(u_{t-1}, \ldots, u_0)_\tau$	τ-adic expansion of $\kappa \in \mathbb{Z}[\tau]$
U_n	first Lucas sequence
V_n	second Lucas sequence
(x, y)	affine point on a Weierstrass curve
x_P, y_P	x-coordinate and y-coordinate of the point P
$(X : Y : Z)$	projective point on a Weierstrass curve
$\mathbb{Z}$	set of integers
$\mathbb{Z}^+$	set of positive integers
$\mathbb{Z}_m$	set of integers modulo m
$\mathbb{Z}_m^*$	set of integers modulo m without 0
$\mathbb{Z}[\tau]$	ring of polynomials in τ with coefficients in $\mathbb{Z}$

Bibliography

[ACS04] Roberto Maria Avanzi, Mathieu Ciet, and Francesco Sica. Faster Scalar Multiplication on Koblitz Curves Combining Point Halving with the Frobenius Endomorphism. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Public Key Cryptography - PKC 2004, 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 28–40. Springer, 2004.

[ADH94] Leonard M. Adleman, Jonathan DeMarrais, and Ming-Deh A. Huang. A subexponential algorithm for discrete logarithms over the rational subgroup of the Jacobians of large genus hyperelliptic curves over finite fields. In Leonard M. Adleman and Ming-Deh A. Huang, editors, *Algorithmic Number Theory, First International Symposium, ANTS-I, Proceedings*, volume 877 of *Lecture Notes in Computer Science*, pages 28–40. Springer, 1994.

[AFLR12] Diego F. Aranha, Armando Faz-Hernández, Julio López, and Francisco Rodríguez-Henríquez. Faster Implementation of Scalar Multiplication on Koblitz Curves. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Proceedings*, volume 7533 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2012.

[Aft11] Alex E. Aftuck. *The Weil Pairing of Elliptic Curves and Its Cryptographic Applications.* University of North Florida, 2011. `http://digitalcommons.unf.edu/etd/139`.

[AHP05] Roberto Maria Avanzi, Clemens Heuberger, and Helmut Prodinger. Minimality of the Hamming Weight of the T-NAF for Koblitz Curves and Improved Combination with Point Halving. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Revised Selected Papers*, volume 3897 of *Lecture Notes in Computer Science*, pages 332–344. Springer, 2005.

[ANS98] ANSI. *ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA).* Accredited Standards Committee X9, 1998.

[ANS01] ANSI. *ANSI X9.63: Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography.* Accredited Standards Committee X9, 2001.

[BBB$^+$12] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. *NIST SP800-57:, Recommendation for Key Management Part 1: General (Revision 3).* National Institute of Standards and Technology, 2012. `http://www.http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf`.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[BF01] Dan Boneh and Matthew K. Franklin. Identity Based Encryption From the Weil Pairing. *IACR Cryptology ePrint Archive*, 2001:90, 2001. `http://eprint.iacr.org/2001/090`.

[BK12] Elaine Barker and John Kelsey. *NIST SP800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators.* National Institute of Standards and Technology, 2012. `http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf`.

[BL14] Daniel J. Bernstein and Tanja Lange. Explicit-Formulas Database. `http://www.hyperelliptic.org/EFD/index.html`, 2014. Online. Last accessed: 2014-04-15.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[BR97] Mihir Bellare and Phillip Rogaway. Minimizing the Use of Random Oracles in Authenticated Encryption Schemes. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *Information and Communication Security, First International Conference, ICICS'97, Proceedings*, volume 1334 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1997.

[Bro06] Daniel R. L. Brown. Conjectured Security of the ANSI-NIST Elliptic Curve RNG. *IACR Cryptology ePrint Archive*, 2006:117, 2006. `http://eprint.iacr.org/2006/117`.

[Cer00] Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters.* Standards for Efficient Cryptography, 2000. Available at `http://www.secg.org/download/aid-386/sec2_final.pdf`.

[Cer09] Certicom Research. *SEC 1: Elliptic Curve Cryptography.* Standards for Efficient Cryptography, May 2009. Available at `http://www.secg.org/download/aid-780/sec1-v2.pdf`.

[CFA$^+$05] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic*

Curve Cryptography. Discrete Mathematics and Its Applications. Taylor & Francis, 2005.

[CLO07] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Number 10 in Undergraduate Texts in Mathematics. Springer, 3rd edition, 2007.

[DF04] David S. Dummit and Richard M. Foote. *Abstract Algebra.* Wiley, 3rd edition, 2004.

[DH76] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[Die07] Claus Diem. Diskrete Mathematik für Informatiker (WS 2007/08). University Lecture Notes, Universität Leipzig. `http://www.math.uni-leipzig.de/~diem/skripten/dm-skript.pdf`, 2007. Online. Last accessed: 2014-05-01.

[Die09] Claus Diem. Lineare Algebra für Mathematiker (WS 2009/10). University Lecture Notes, Universität Leipzig. `http://www.math.uni-leipzig.de/~diem/skripten/la-mathe-skript.pdf`, 2009. Online. Last accessed: 2014-05-01.

[Die12] Claus Diem. What on earth is "index calculus"? `http://ellipticnews.wordpress.com/2012/05/07/246/`, May 2012. Online. Last accessed: 2014-01-16.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Eng99] Andreas Enge. *Elliptic Curves and Their Applications to Cryptography - an Introduction.* Kluwer, 1999.

[Flo67] Robert W. Floyd. Nondeterministic Algorithms. *Journal of the ACM*, 14(4):636–644, October 1967.

[Gal14] Steven Galbraith. ellipticnews - New discrete logarithm records, and the death of Type 1 pairings. `https://ellipticnews.wordpress.com/2014/02/01/new-discrete-logarithm-records-and-the-death-of-type-1-pairings/`, February 2014. Online. Last accessed: 2014-04-25.

[GLV00] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Improving the parallelized Pollard lambda search on anomalous binary curves. *Mathematics of Computation*, 69(232):1699–1705, 2000.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[Han10] Christian Hanser. New Trends in Elliptic Curve Cryptography. Master's Thesis, Graz University of Technology, Institute for Applied Information Processing and Communications, Graz University of Technology, April 2010.

[Her96] Israel N. Herstein. *Abstract Algebra.* Prentice Hall, 3rd edition, 1996.

[HMV04] Darrel Hankerson, Alfred J. Menezes, and Scott A. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer Professional Computing. Springer, 2004.

[HT00] Akira Higuchi and Naofumi Takagi. A fast addition algorithm for elliptic curve arithmetic in GF(2^n) using projective coordinates. *Information Processing Letters*, 76(3):101–103, 2000.

[HW13] Christian Hanser and Christian Wagner. Speeding Up the Fixed-Base Comb Method for Faster Scalar Multiplication on Koblitz Curves. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, Lida Xu, Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Proceedings*, volume 8128 of *Lecture Notes in Computer Science*, pages 168 – 179. Springer, 2013.

[IEE00] IEEE Std 1363-2000. *IEEE Standard Specifications for Public-Key Cryptography.* IEEE, New York, 2000.

[ISO98] ISO/IEC. *ISO/IEC 14888-3: Information Technology - Security Techniques - Digital signatures with appendix - Part 3: Discrete logarithm based mechanisms.* International Organization for Standardization / International Electrotechnical Commission, 1998.

[ISO02] ISO/IEC. *ISO/IEC 15946-3: Information Technology - Security Techniques Cryptographic Techniques Based on Elliptic Curves Part 3: Key Establishment.* International Organization for Standardization / International Electrotechnical Commission, 2002.

[ISO06] ISO/IEC. *ISO/IEC 18033-2: Information Technology - Security Techniques Encryption Algorithms Part 2: Asymmetric Ciphers.* International Organization for Standardization / International Electrotechnical Commission, 2006.

[IT88] Toshiya Itoh and Shigeo Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases. *Information and Computation*, 78(3):171–177, 1988.

[JMV01] Don Johnson, Alfred J. Menezes, and Scott A. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.

[Joh97] Don Johnson. "Key Validation". *Contribution to ANSI X9F1 working group*, 1997.

[Jou00] Antoine Joux. A One Round Protocol for Tripartite Diffie-Hellman. In Wieb Bosma, editor, *Algorithmic Number Theory, 4th International Symposium, ANTS-IV,Proceedings*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000.

[Jou13a] Antoine Joux. A new index calculus algorithm with complexity $L(1/4 + o(1))$ in very small characteristic. *IACR Cryptology ePrint Archive*, 2013:95, 2013. `http://eprint.iacr.org/2013/095`.

[Jou13b] Antoine Joux. Discrete Logarithms in $GF(2^{6168})$ [$= GF((2^{257})^{24})$]. `https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;49bb494e.1305`, May 2013. Online. Last accessed: 2014-04-25.

[Knu69] Donald E. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms. Addison-Wesley Publishing Company, Reading, Massachusetts, 1969. exercises 6 and 7, page 7.

[KO62] Anatolii A. Karatsuba and Yuri Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. In *Doklady Akademii Nauk SSSR*, volume 145, pages 293–294, 1962. (Translation in Physics-Doklady 7, pages 595–596, 1963).

[Kob87] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[Kob89] Neal Koblitz. Hyperelliptic Cryptosystems. *Journal of Cryptology*, 1(3):139–150, 1989.

[Kob91] Neal Koblitz. CM-Curves with Good Cryptographic Properties. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 279–287. Springer, 1991.

[KS01] Fabian Kuhn and René Struik. Random Walks Revisited: Extensions of Pollard's Rho Algorithm for Computing Multiple Discrete Logarithms. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Revised Papers*, volume 2259 of *Lecture Notes in Computer Science*, pages 212–229. Springer, 2001.

[Lan04] Tanja Lange. A note on López-Dahab coordinates. *IACR Cryptology ePrint Archive*, 2004:323, 2004. `http://eprint.iacr.org/2004/323`.

[LD98] Julio López and Ricardo Dahab. Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n). In Tavares and Meijer [TM99], pages 201–212.

[LD00] Julio López and Ricardo Dahab. High-Speed Software Multiplication in F_{2^m}. In Bimal K. Roy and Eiji Okamoto, editors, *Progress in Cryptology - INDOCRYPT 2000, First International Conference in Cryptology in India, Proceedings*, volume 1977 of *Lecture Notes in Computer Science*, pages 203–212. Springer, 2000.

[LL94] Chae Hoon Lim and Pil Joong Lee. More Flexible Exponentiation with Precomputation. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94,14th Annual International Cryptology Conference, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 1994.

[LM10] M. Lochter and J. Merkle. Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation, 2010.

[LMQ$^+$03] Laurie Law, Alfred J. Menezes, Minghua Qu, Jerome Solinas, and Scott A. Vanstone. An Efficient Protocol for Authenticated Key Agreement. *Design, Codes and Cryptography*, 28(2):119–134, March 2003.

[MHH12] Nashwa A. F. Mohamed, Mohsin H. A. Hashim, and Michael Hutter. Improved Fixed-Base Comb Method for Fast Scalar Multiplication. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Proceedings*, volume 7374 of *Lecture Notes in Computer Science*, pages 342–359. Springer, 2012.

[Mil85] Victor S. Miller. Use of Elliptic Curves in Cryptography. In Hugh C. Williams, editor, *Advances in Cryptology - CRYPTO '85,Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[Mil04] Victor S. Miller. The Weil Pairing, and Its Efficient Calculation. *Journal of Cryptology*, 17(4):235–261, September 2004.

[MOV91] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 80–89. ACM, 1991.

[MQV95] Alfred J. Menezes, Minghuan Qu, and Scott A. Vanstone. Some new key agreement protocols providing implicit authentication. In *Workshop Record, 2nd Workshop on Selected Areas in Cryptography (SAC'95)*, pages 22–32, 1995.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., 1st edition, 1996.

[Nat00] National Institute of Standards and Technology. *FIPS PUB 186-2: Digital Signature Standard (DSS).* National Institute of Standards and Technology, January 2000. Available at `http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf`.

[Nat09] National Institute of Standards and Technology. *FIPS PUB 186-3: Digital Signature Standard (DSS).* National Institute of Standards and Technology, June 2009. Available at `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.

[Nat13] National Institute of Standards and Technology. *FIPS PUB 186-4: Digital Signature Standard (DSS).* National Institute of Standards and Technology, 2013. Available at `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`.

[OW99] Paul C. Van Oorschot and Michael J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12:1–28, 1999.

[Pol75] John M. Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

[Pol78] John M. Pollard. Monte Carlo methods for Index Computation (mod p). *Mathematics of Computation*, 32:918–924, 1978.

[Rib91] Paulo Ribenboim. *The Little Book of Big Primes.* Springer, 1991.

[Rib00] Paulo Ribenboim. *My Numbers, My Friends: Popular Lectures on Number Theory.* Springer, 2000.

[SA98] Takakazu Satoh and Kiyomichi Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. *Commentarii Mathematici Universitatis Sancti Pauli*, 47:81–92, 1998.

[Sah12] Chandan Saha. Computational Number Theory and Algebra - Lecture 21 - The Index Calculus method. University Lecture Notes, Max-Planck-Institut für Informatik. `http://www.mpi-inf.mpg.de/~csaha/lectures/lec21.pdf`, 2012. Online. Last accessed: 2014-02-11.

[SEB09] Augustin P. Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard. A Secure and Efficient Authenticated Diffie-Hellman Protocol. In Fabio Martinelli and Bart Preneel, editors, *Public Key Infrastructures, Services and Applications - 6th European Workshop, EuroPKI 2009, Revised Selected Papers*, volume 6391 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2009.

[Sem98] Igor A. Semaev. Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. *Mathematics of Computation*, 67(221):353–356, 1998.

[Sil09] Joseph H. Silverman. *The Arithmetic of Elliptic Curves.* Graduate Texts in Mathematics. Springer, 2nd edition, 2009.

[Sma99] Nigel P. Smart. The Discrete Logarithm Problem On Elliptic Curves Of Trace One. *Journal of Cryptology*, 12:193–196, 1999.

[Sma01] Nigel P. Smart. An Identity Based Authenticated Key Agreement Protocol Based on the Weil Pairing. *IACR Cryptology ePrint Archive*, 2001:111, 2001. `http://eprint.iacr.org/2001/111`.

[Sma12] Nigel P. Smart. *Cryptography: An Introduction.* Ebook, 3rd edition, June 2012. `http://www.cs.bris.ac.uk/~nigel/Crypto_Book/`.

[Sol00] Jerome A. Solinas. Efficient Arithmetic on Koblitz curves. *Designs, Codes, and Cryptography*, 19(2/3):195–249, 2000.

[SOOS95] Richard Schroeppel, Hilarie K. Orman, Sean W. O'Malley, and Oliver Spatscheck. Fast Key Exchange with Elliptic Curve Systems. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 1995.

[SS06] Berry Schoenmakers and Andrey Sidorenko. Cryptanalysis of the Dual Elliptic Curve Pseudorandom Generator, 2006. `http://eprint.iacr.org/2006/190`.

[Str64] Ernst G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70:806–808, 1964.

[TC05] Woei-Jiunn Tsaur and Chih-Ho Chou. Efficient algorithms for speeding up the computations of elliptic curve cryptosystems. *Applied Mathematics and Computation*, 168(2):1045 – 1064, 2005.

[Tes00] Edlyn Teske. On Random Walks For Pollard's Rho Method. *Mathematics of Computation*, 70:809–825, 2000.

[TM99] Stafford E. Tavares and Henk Meijer, editors. *Selected Areas in Cryptography '98, SAC'98, Proceedings*, volume 1556 of *Lecture Notes in Computer Science*. Springer, 1999.

[Van92] Scott A. Vanstone. Responses to NIST's proposal. *Communications of the ACM*, 35(7):50–52, July 1992.

[Wal98] David A.R. Wallace. *Groups, Rings, and Fields.* Springer Undergraduate Mathematics Series. Springer, 1998.

[WZ98] Michael J. Wiener and Robert J. Zuccherato. Faster Attacks on Elliptic Curve Cryptosystems. In Tavares and Meijer [TM99], pages 190–200.

[YISK11] Masaya Yasuda, Tetsuya Izu, Takeshi Shimoyama, and Jun Kogure. On random walks of Pollard's rho method for the ECDLP on Koblitz curves. *Journal of Math-for-industry*, 3:107–112, 2011.